68' MICRO JOURNAL

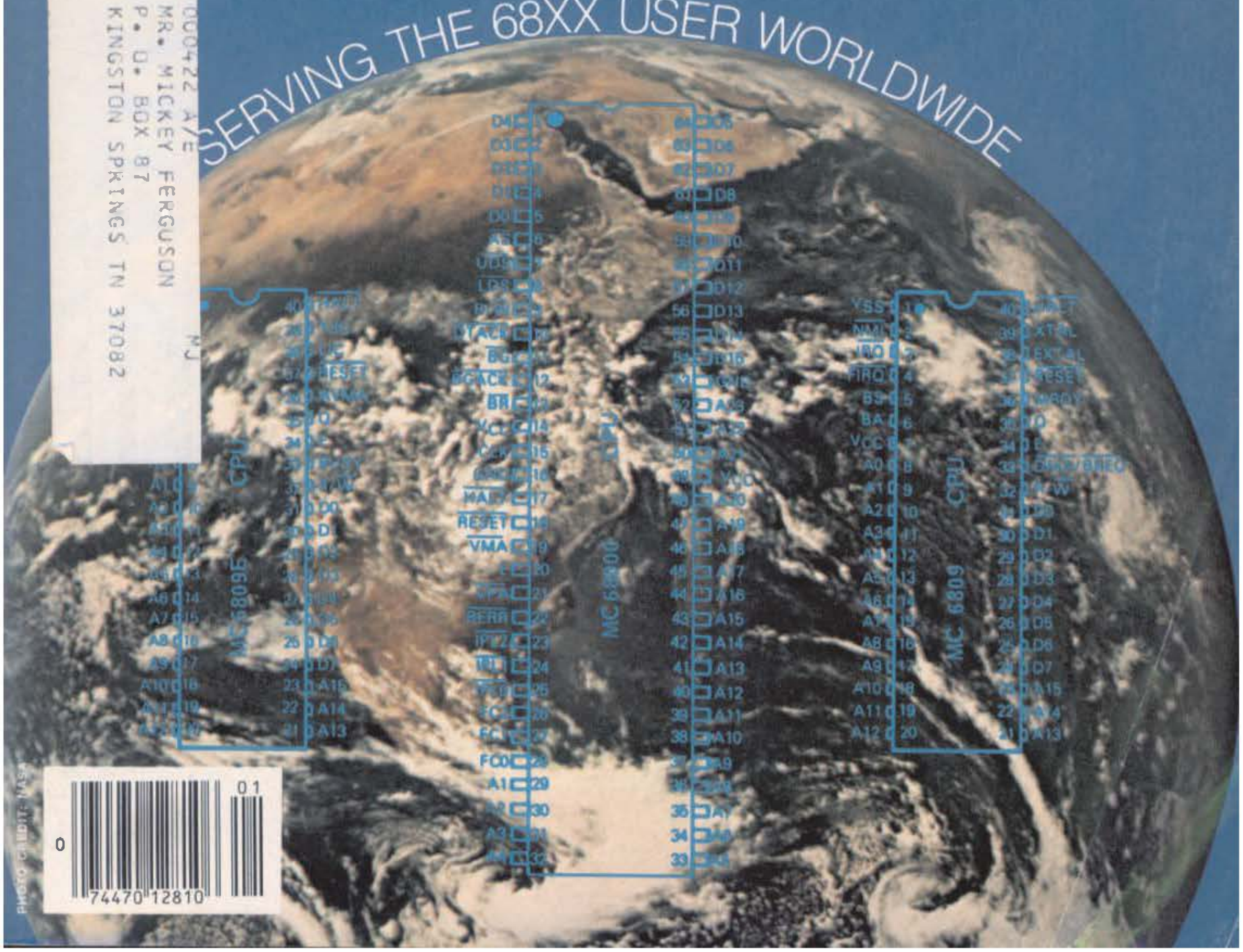
Australia A $4.75 New Zealand NZ $6.50
Singapore S $9.45 Hong Kong H $23.50
Malaysia M $9.45 Sweden 30:-SEK

$2.95 USA

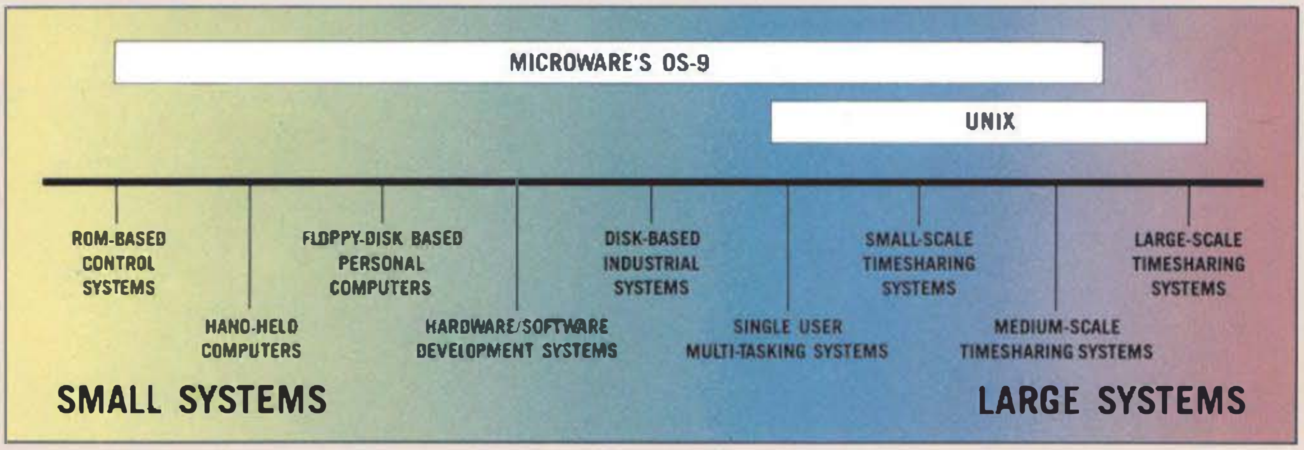
VOLUME VIII ISSUE I • Devoted to the 68XX User • January 1986

"Small Computers Doing Big Things"

000422 A/E MJ
MR. MICKEY FERGUSON
P. O. BOX 87
KINGSTON SPRINGS TN 37082



Only Microware's OS-9 Operating System Covers the Entire 68000 Spectrum


MICROWARE'S OS-9
UNIX
ROM-BASED CONTROL SYSTEMS
HAND-HELD COMPUTERS
FLOPPY-DISK BASED PERSONAL COMPUTERS
HARDWARE/SOFTWARE DEVELOPMENT SYSTEMS
DISK-BASED INDUSTRIAL SYSTEMS
SINGLE USER MULTI-TASKING SYSTEMS
SMALL-SCALE TIMESHARING SYSTEMS
MEDIUM-SCALE TIMESHARING SYSTEMS
LARGE-SCALE TIMESHARING SYSTEMS
SMALL SYSTEMS
LARGE SYSTEMS

Is complicated software and expensive hardware keeping you back from Unix? Look into OS-9, the operating system from Microware that gives 68000 systems a Unix-style environment with much less overhead and complexity.

OS-9 is versatile, inexpensive, and delivers outstanding performance on any size system. The OS-9 executive is much smaller and far more efficient than Unix because it's written in fast, compact assembly language, making it ideal for critical real-time applications. OS-9 can run on a broad range of 8 to 32 bit systems based on the 68000 or 6809 family MPUs from ROM-based industrial controllers up to large multiuser systems.

OS-9'S OUTSTANDING C COMPILER IS YOUR BRIDGE TO UNIX

Microware's C compiler technology is another OS-9 advantage. The compiler produces extremely fast, compact, and ROMable code. You can easily develop and port system or application software back and forth to standard Unix systems. Cross-compiler versions for VAX and PDP-11 make coordinated Unix/OS-9 software development a pleasure.

SUPPORT FOR MODULAR SOFTWARE — AN OS-9 EXCLUSIVE

Comprehensive support for modular software puts OS-9 a generation ahead of other operating systems. It multiplies programmer productivity and memory efficiency. Application software can be built from individually testable software modules including standard "library" modules. The modular structure lets you customize and reconfigure OS-9 for specific hardware easily and quickly.

A SYSTEM WITH A PROVEN TRACK RECORD

Once an underground classic, OS-9 is now a solid hit. Since 1980 OS-9 has been ported to over a hundred 6809 and 68000 systems under license to some of the biggest names in the business. OS-9 has been imbedded in numerous consumer, industrial, and OEM products, and is supported by many independent software suppliers.

Key OS-9 Features At A Glance

- Compact (16K) ROMable executive written in assembly language
- User "shell" and complete utility set written in C
- C-source code level compatibility with Unix
- Full Multitasking/multiuser capabilities
- Modular design - extremely easy to adapt, modify, or expand
- Unix-type tree structured file system
- Rugged "crash-proof" file structure with record locking
- Works well with floppy disk or ROM-based systems
- Uses hardware or software memory management
- High performance C, Pascal, Basic and Cobol compilers

microware® OS-9

AUTHORIZED MICROWARE DISTRIBUTORS

AUSTRALIA
MICROPROCESSOR CONSULTANTS
16 Bandera Avenue
Wagga Wagga 2650
NSW Australia
phone: 616-931-2331

ENGLAND
VIVAWAY LTD.
36-38 John Street
Luton, Bedfordshire
England LU1 2JE
phone: (0582) 423425
telex: 825115

JAPAN
MICROWARE JAPAN LTD.
3-8-9 Baraki, Ichikawa
Chiba 272-01, Japan
phone: 0473 (28) 4493
telex: 781-299-3122

SWEDEN
MICROMASTER SCANDINAVIAN AB
St Persgatan 7
Box 1309
S751-43 Uppsala
Sweden
phone: 018-138595
telex: 76129

SWITZERLAND
ELSOFT AG
Bankstrasse 9
5432 Neuenhof
Switzerland
phone: (41) 056-862724
telex: 57136

USA
MICROWARE SYSTEMS CORPORATION
1866 NW 114th Street
Des Moines, Iowa 50322
USA
phone: 515-224-1929
telex: 910-520-2535
FAX: 515-224-1352

WEST GERMANY
DR. KEIL, GMBH
Porphystrasse 15
D-6905 Schriesheim
West Germany
phone: (0 62 03) 67 41
telex: 465025

OS-9 is a trademark of Microware and Motorola. Unix is a trademark of Bell Labs.

ALL SYSTEMS INCLUDE:

- The CLASSY CHASSIS with a ferro-resonant, constant voltage power supply that provides +8 volts at 30 Amps, + 16 volts at 5 Amps, and − 16 volts at 5 Amps.
- Gold plated bus connectors.
- Double density DMA floppy disk controllers.
- Complete hardware and software documentation.
- Necessary cables, filler plates.

YOU CAN EXPAND YOUR SYSTEM WITH:

MASS STORAGE

Dual 8'' DSDD Floppies, Cabinet & Power Supply	$1698.88
60MB Streamer (UniFLEX-020 only)	$2400.00
1.6MB Dual Speed Floppy	(under development)

MEMORY

#67 Static RAM-64K NMOS (6809 Only)	$349.67
#64 Static RAM-64K CMOS w/battery (6809 Only)	$398.64
#72 256K CMOS Static RAM w/battery	$998.72
#31 16 Socket PROM/ROM/RAM Board (6809 only)	$268.31

INTELLIGENT I/O PROCESSOR BOARDS

significantly reduce systems overhead by handling routine I/O functions; freeing the host CPU for running user programs. This improves overall system performance and allows user terminals to be run at up to 19.2K baud. For use with GMX III and 020 systems.

#11 3 Port Serial-30 Pin (OS9)	$498.11
#14 3 Port Serial-30 Pin (UniFLEX)	$498.14
#12 Parallel-50 Pin (UniFLEX-020)	$538.12
#13 4 Port Serial-50 Pin (OS9 & UniFLEX-020)	$618.13

I/O BOARDS (6809 SYSTEMS ONLY)

#41 Serial, 1 Port	$88.41
#43 Serial, 2 Port	$128.43
#46 Serial, 8 Port (OS9/FLEX only)	$318.46
#42 Parallel, 2 Port	$88.42
#44 Parallel, 2 Port (Centronics pinout)	$128.44
#45 Parallel, 8 Port (OS9/FLEX only)	$198.45

CABLES FOR I/O BOARDS—SPECIFY BOARD

#95 Cable sets (1 needed per port)	$24.95
#51 Cent. B.P. Cable for #12 & #44	$34.51
#53 Cent. Cable Set	$36.53

OTHER BOARDS

#66 Prototyping Board-50 Pin	$56.66
#33 Prototyping Board-30 Pin	$38.33
Windrush EPROM Programmer S30 (OS9/FLEX 6809 only)	$545.00

CONTACT GIMIX FOR FURTHER DETAILS ON THESE AND OTHER BOARDS AND OPTIONS.

EXPORT MODELS: ADD $30 FOR 50Hz. POWER SUPPLIES.
ALL PRICES ARE F.O.B. CHICAGO.

GIMIX DOES NOT GUARANTEE PERFORMANCE OF ANY GIMIX SYSTEMS, BOARDS OR SOFTWARE WHEN USED WITH OTHER MANUFACTURERS PRODUCT.

GIMIX, Inc. reserves the right to change pricing, terms, and products specifications at any time without further notice.

GIMIX 2MHZ 6809 SYSTEMS / GMX 68020 SYSTEMS

	GIMIX 2MHZ 6809 SYSTEMS					GMX 68020 SYSTEMS
Operating Systems Included	#49 OS9 GMX I/ and FLEX	#39 OS9 GMX II/ and FLEX	#79 OS9 GMX III/ and FLEX	#39 UniFLEX	#89 UniFLEX III	#020 UniFLEX VM
CPU Included	#05	#05	GMX III	#05	GMX III	GMX 020 + MMU
Serial Ports Included	2	2	3 Intelligent	2	3 Intelligent	3 Intelligent
High Speed Static RAM	64KB	256KB	256KB	256KB	256KB	1 Megabyte
PRICES OF SYSTEMS WITH: Dual 80 Track DSDD Drives	$2998.49	$3398.39	$4898.79	N/A	N/A	N/A
25MB Hard Disk and one 80 track Floppy Disk	$5598.49	$5998.39	$7798.79	$5998.39	$8098.89	$13,680.20
72 MB Hard Disk and one 80 track	$7598.49	$7998.39	$9798.79	$7998.39	$10,098.89	$15,680.20
a 72MB + a 6MB removable pack hard disk and one 80 track floppy	$9098.49	$9498.30	N/A	$9498.39	N/A	N/A
a 72MB + a 12MB removable pack hard disk and one 80 track floppy	N/A	N/A	$11,298.79	N/A	$11,598.89	$17,180.20

GMX 6809 OS9/ FLEX SYSTEMS SOFTWARE			
OS9 + Editor, Assembler, Debugger	GMX I Included	GMX II Included	GMX III Included
FLEX	Included	Included	Included
GMXBUG Monitor	Included	Included	Included
Basic 09, RunB (OS9)	Included	Included	Included
RMS (OS9)	Included	Included	Included
DO (OS9)	Included	Included	Included
VDisk for FLEX	N/A	Included	Included
RAMDisk for OS9	N/A	$125 option	Included
O-FLEX	N/A	$250 option	Included
Support ROM	N/A	N/A	Included
Hardware CRC	N/A	N/A	Included

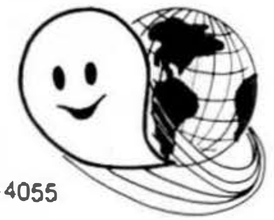
Available: Wide variety of languages and other software for use with either OS-9 or FLEX.

All GIMIX versions of OS9 can read and write RS color computer format OS9 disks, as well as the Microware/GIMIX standard format.

All OS9/FLEX systems allow you to software select either operating system.

TO ORDER BY MAIL: SEND CHECK OR MONEY ORDER OR USE YOUR VISA OR MASTER CHARGE. Please allow 3 weeks for personal checks to clear. U.S. orders add $5 handling if order is under $200.00. Foreign orders add $10 handling if order is under $200.00. Foreign orders over $200.00 will be shipped via Emery Air Freight COLLECT, and we will charge no handling. All orders must be prepaid in U.S. funds. Please note that foreign checks have been taking about 8 weeks for collection so we would advise wiring money, or checks drawn on a bank account in the U.S. Our bank is the Continental Illinois National Bank of Chicago, 231 S. LaSalle Street, Chicago, IL 60693, account number 73-32033.

GIMIX inc.



1337 WEST 37th PLACE
CHICAGO, ILLINOIS 60609
(312) 927-5510 • TWX 910-221-4055

'68' MICRO JOURNAL

Portions of the text for '68' Micro Journal were prepared using the following furnished Hard/Software:

COMPUTERS – HARDWARE

Southwest Technical Products
219 W. Rhapsody
San Antonio, TX 78216
S, 9 - 5/8 DMF Disk - CDS1 - 8212W - Sprint 3 Printer

—

GIMIX Inc.
1337 West 37th Place
Chicago, IL 60609
Super Mainframe - OS9 - FLEX - Assorted Hardware

EDITORS – WORD PROCESSORS

Technical Systems Consultants, Inc.
111 Providence Road
Chapel Hill, NC 27514
FLEX - Editor - Text Processor

—

Stylo Software Inc.
PO Box 916
Idaho Falls, ID 83402
Stylograph - Mail Merge - Spell

Editorial Staff

Don Williams Sr. .... Publisher
Larry E. Williams .... Executive Editor
Tom E. Williams .... Production Editor
Robert L. Nay .... Technical Editor

Administrative Staff

Mary Robertson .... Officer Manager
Joyce Williams .... Subscriptions
Christine Lea .... Accounting

Contributing Editors

Ron Anderson	Norm Commo
Peter Dibble	William E. Fisher
Dr. Theo Elbert	Carl Mann
Dr. E.M. Pass	Ron Voights
Philip Lucido	Randy Lewis

Special Technical Projects

Clay Abrams K6AEP
Tom Hunt

Contents

Vol. VIII, Issue 1 — January 86

Send All Correspondence To:

Computer Publishing Center
68' Micro Journal
5900 Cassandra Smith Rd.
Hixson, TN 37343

Phone (615) 842-4600 or Telex 5106006630

Copyrighted 1985 by Computer Publishing Inc.

68' Micro Journal is published 12 times a year by Computer Publishing Inc. Second Class Postage Paid ISSN 0194-5025 at Hixson, TN and additional entries. Postmaster: send form 3597 to 68' Micro Journal, POB 849 Hixson, TN 37343.

Subscription Rates

1 Year $24.50 U.S.A., Canada & Mexico Add **$9.50** a Year. Other **Foreign Add $12 a Year** for **Surface, Airmail Add $48 a Year. Must be in U.S. currency!!**

Items or Articles For Publication

Articles submitted for publication should include authors name, address, telephone number and date. Articles should be on **either 5 or 8 inch disk** in STYLOGRAPH or TSC Editor format with **3.5 inch column width.** All disks will be returned. Articles submitted on **paper** should be **4.5 inches in width** (including Source Listings) for proper reductions. **Please Use A Dark Ribbon!!** No Blue Ink!!! Single space on 8x11 bond or better grade paper. No hand written articles accepted. Disks should be in FLEX2 6800 or FLEX9 6809 any version or OS-9 any version.

The following TSC Text Processor commands **ONLY** should be used: .sp space, .pp paragraph, .fi fill and .nf no fill. Also please **do not format** within the text with **multiple spaces.** We will enter the rest at time of editing.

All STYLOGRAPH commands are acceptable except .pg page command. We print edited text files in continous text form.

Letters To The Editor

All letters to the editor should comply with the above requirements and **must be signed.** Letters of "gripes" as well as "praise" are solicited. We reserve the right to reject any submission for lack of "good taste" and we reserve the right to define "good taste".

Advertising Rates

Commercial advertisers please contact 68' Micro Journal advertising department for current rate sheet and requirements.

Classified Advertising

All classified ads must be non-commercial. Minimum of **$9.50 for first 20 words** and .45 per word after 20. All classifieds must be **paid in advance.** No classified ads accepted over the phone.

..HEAR YE.....HEAR OS-9™ User Notes

By: Peter Dibble
As Published in 68 Micro Journal

The publishers of 68 Micro Journal are proud to announce the publication of Peter Dibbles OS9 USER NOTES.

Information for the BEGINNER to the PRO, Regular or CoCo OS9

Using OS9
HELP, HINTS, PROBLEMS, REVIEWS, SUGGESTIONS, COMPLAINTS, OS9 STANDARDS, Generating a New Bootstrap, Building a new System Disk, OS9 Users Group, etc.

Program interfacing to OS9
DEVICE DESCRIPTORS, DIRECTORIES, "FORKS", PROTECTION, "SUSPEND STATE", "PIPES", "INPUT/OUTPUT SYSTEM", etc.

Programming Languages
Assembly Language Programs and Interfacing; Basic09, C, Pascal, and Cobol reviews, programs, and uses; etc.

Disks Include
No typing all the Source Listings in. Source Code and, where applicable, assembled or compiled **Operating Programs.** The Source and the Discussions in the Columns can be used "as is", or as a "Starting Point" for developing **your** OWN more powerful Programs. Programs sometimes use multiple Languages such as a short Assembly Language Routine for reading a Directory, which is then "piped" to a Basic09 Routine for output formatting, etc.

BOOK Typeset -- w/ Source Listings (3-Hole Punched; 8 x 11) **$9.95**

Deluxe Binder - - - - - - - - - - - $5.50

All Source Listings on Disk
1 8" SS, SD Disk - - - - $14.95
2 5" SS, DD Disks - - - $24.95

Shipping and Handling; $3.50 per **Book**, $2.50 per **Disk** Set
* All Currency in U.S. Dollars
Foreign Orders Add $4.50 S/H (Surface) or $7.00 S/H (Air Mail)
If paying by check - Please allow 4-6 weeks delivery

Continually Updated In 68 Micro Journal Monthly

Computer Publishing Inc.
5900 Cassandra Smith Rd.
Hixson, TN. 37343

VISA

(615) 842-4600
Telex 5106006630

TM - OS9 is a trademark of Microware Systems Corp. and Motorola Inc.
TM - 68' Micro Journal is a trademark of Computer Publishing Inc.

MUSTANG-020 Super SBC™

***

We Proudly Announce the MUSTANG-020 Super SBC*
"The one with the **REAL KICK!**"
Only from DATA-COMP

This is the NCC, world beater GMX SBC, in a super configuration. Data-Comp has mated it to a power plus power supply/stylish cabinet and your choice of floppy and/or hard disk drives. Available in several different configurations. (1) single board. (2) single board and regulators for your cabinet or mainframe and power supply. (3) single board - power supply and cabinet - your disk drives. (4) single board - power supply/cabinet - our drives configured to your specs, and ready to run. OS9 68K will be available as well as several other popular operating systems. Also all the popular OS9 68K software and Motorola 020-BUG will be available at a very reasonable price.

Note: We will include the Motorola 020-BUG at no additional charge. This item alone sells for $500.00. This allows direct coding of 68020 advanced codes. 020-BUG is required for our version of OS9. Please bear in mind, this system is the one others are compared to.

This system is the state-of-the-art star-ship. It runs **rings** around any other 68XXX SBC, and most mainframes. The speed and expanded RAM make this the "best buy" by a far stretch! A true multi-user, multi-tasking computer. So far advanced that even the experts don't call it a micro. Compared to the others, it isn't even a "horse race." And the price is certainly right. You can bet on this one!

So, will it be **Turtle** or **Thoroughbred?**

MUSTANG-020 System Prices
Effective November 1985

Mustang-020 SBC, wired & tested with 4 DB25 Serial ports pre-wired, ready to install with your cabinet, P/S, CRT and drives	$2750.00
M020 Cabinet and P/S, for Mustang-020, less cables	$269.95
M020 Cables, dual floppy or winchester, specify which - floppy or winchester	$39.95
M020FC Floppy cabinet and P/S, holds and powers 2 thin-line floppies	$79.95
M020F Floppy, 80 track, DD/DS	$269.95
OS-9, SPECIAL Mustang-020 version	$350.00
MC68881F/P co-processor	$495.00
10 Megabyte Winchester	$695.00
20 Megabyte Winchester	$895.00
Winchester Controller (Xebec-OMTI)	$395.00

Note: for orders of complete systems (Mustang-020, cabinet & P/S, disk drives and OS-9, deduct 5% from total package. (limited time offer) **See opposite page.**

**** Special Winchester Notice ****

The Mustang-020 device descriptors will allow you to use practically **ANY** winchester drive supported by XEBEC or OMTI controllers.

. . .

Include: $3.50 SBC, cables only S/H. Cabinets include $7.50 S/H. Complete System include $20.00. All checks must be in USA funds. Overseas specify shipping instructions and sufficient funds.

* Mustang-020 is trademark of Data-Comp-CPI

DATA-COMP
5900 Cassandra Smith Rd.
Hixson, TN 37343

VISA

SHIPPING
USA ADD 2%
FOREIGN ADD 5%
MIN. $2.50

(615)842-4600
For Ordering
TELEX 5106006630

MUSTANG-020 Super SBC ™

Mustang 020 Features

- 12.5 MHz MC68020 full 32-Bit wide path Processor
  - 32-bit wide non-multiplexed data & address buses
  - On-chip instruction cache
  - Object-code compatible with earlier M68000 family processors (68000/68008/68010)
  - Enhanced instruction set - Coprocessor interface
- Optional 68881 Floating point Coprocessor (12.5 MHz)
  - Direct extension of 68020 instruction set
  - Full support of IEEE P754, draft 10.0
  - Transcendentals and other math functions
- 2 Megabytes of RAM (512K x 32-bit organization)
- Up to 256K bytes of EPROM (64K X 32-bits)
  - Uses four 2764, 27128, 27256, or 27512 EPROMs
- 4 Asynchronous serial I/O ports (2 x MC68681 DUART)
  - Software programmable baud rates to 19.2K
  - Standard RS-232 interface
  - Optional network interface on one port
- Buffered 8-bit Parallel I/O Port (1/2 MC68230)
  - Centronics-type parallel printer pinout
  - May also be used as parallel input port
- Expansion Connector for Additional I/O Devices
  - 16-bit data path
  - 256 byte address space
  - 2 interrupt inputs
  - Clock and Control Signals
- Time-of-Day Clock/Calendar w/battery backup
- Controller for up to Two 5 1/4" Floppy Disk Drives
  - Single or double sided
  - Single or double density
  - 48 or 96 tracks per inch (40/80 Track)
- Mounts Directly to a Standard 5 1/4" Disk Drive
- SASI interface for Intelligent Hard Disk Controllers
- Programmable Periodic Interrupt Generator
  - For time-slicing and real-time applications
  - Interrupt rates from microseconds to seconds
  - Highly Accurate timebase (5 PPM)
- 5-bit sense switch, readable by the processor
- Hardware single-step capability

*** And many other advanced features not found on other systems priced much higher.

Load Basic09:

Tested Benchmarks:	RAM Disk	Floppy	Winchester
Regular 68XXX	3.0 secs	Ukn	7 secs
MUSTANG - 020	0.3 secs	3 secs	0.9 secs

MUSTANG-020 Benchmarks
Time Seconds

Type System	32 bit Int. Loop	Register Long Loop
IBM AT 7300 Xenix Sys 3	9.7	No Registers
AT&T 7300 UNIX PC 68010	7.2	4.3
DEC VAX 11/780 UNIX Berkley 4.2	3.6	3.2
DEC VAX 11/750 " " "	5.1	3.2
68008 OS9 68K 8 Mhz	18.0	9.0
68000 " " 10 Mhz	6.5	4.0
MUSTANG-020 68020 MC68881 OS9 16 Mhz	2.2	0.88
MUSTANG-020 68020 MC68881 UNIFLEX "	1.8	1.22

```
C Loop - For I - 1 to 999,999
         I++
```

Estimated MIP - MUSTANG-020 - 2.5 MIP
Motorola Specs: Burst up to 7 MIP - 16 Mhz

Note: The standard MUSTANG-020 is now available with the TSC UniFLEX System. TSC software is optimized for the 68020 and MC68881 Co-Processor. This includes the TSC "C" compiler and other software. Speeds are greatly enhanced!

(615)842-4600
Telex 5106006630

DATA-COMP

For a limited time we will offer $400 Trade-In on your old Q--- 68008 or 68000 SBC. must be working properly and complete with all software and cables. Call for more information!

Standard Systems
MUSTANG-020 Prices (Nov 13.1985) Net

	Dual 5" 80 trk. Floppy No Winchester		Winchester & 1 Floppy
020 Board	$2750.00		$2750.00
Cabinet	269.95		269.95
5"-80 trk Floppy(2)	539.90	(1)	269.95
Floppy cable	39.95		39.95
OS-9 68K	350.00		350.00
		Winchester cable	39.95
Total System	$3949.80	Winchester controller	395.00
Less 5%	-197.49	10 MegByte Winchester	695.00
	$3752.31	Total System	$4809.80
S/H UPS	20.00	Less 5%	-240.49
Total	$3772.31		$4569.31
		S/H UPS	20.00
		Total	$4589.31
		With 20 MegByte Winchester Add:	200.00
			$4789.31

NOTE: THIS IS A COMPLETE TWO (2) MEGABYTE, FOUR (4) USER SYSTEM. SOON EXPANDABLE TO 12 USER.

ALSO UniFLEX WILL BE AVAILABLE FOR THIS SYSTEM SOON.

ALSO INTELLIGENT I/O

PLEASE NOTE THE VERIFIED BENCHMARKS ABOVE.

WE BRING YOU "TOOLS" NOT "TOYS"!

NOTE: 6881 Co-Processor Add $495.00

More help than any other thing I have ever read, except 68 Micro Journal, of course. Come on with the C book on Norm Commo's column.
BD

FLEX™ USER NOTES
THE 6800-6809 BOOK

By: Ronald W. Anderson
As published in 68 MICRO JOURNAL™

SWELL! - Best buy EVER in a 6800/6809 book.
JTW

The publishers of 68 MICRO JOURNAL are proud to announce the publication of Ron Anderson's **FLEX USER NOTES**, in book form. This popular monthly column has been a regular feature in 68 MICRO JOURNAL SINCE 1979. It has earned the respect of thousands of 68 MICRO JOURNAL readers over the years. In fact, Ron's column has been described as the 'Bible' for 68XX users, by some of the world's leading microprocessor professionals. Now all his columns are being published, in whole, as the most needed and popular 68XX book available. Over the years Ron's column has been one of the most popular in 68 MICRO JOURNAL. And of course 68 MICRO JOURNAL is the most popular 68XX magazine published.

As a SPECIAL BONUS all the source listing in the book will be available on disk for the low price of: FLEX™ format only — 5" $12.95 — 8" $16.95 plus $2.50 shipping and handling, if ordered with the book. If ordered separately the price of the disks will be: 5" $17.95 — 8" $19.95 plus $2.50 shipping and handling.

Listed below are a few of the **TEXT** files included in the book and on diskette.

All TEXT files in the book are on the disks.

LOGO.C1	File load program to offset memory — ASM PIC
MEMOVE.C1	Memory move program — ASM PIC
DUMP.C1	Printer dump program — uses LOGO — ASM PIC
SUBTEST.C1	Simulation of 6800 code to 6809, show differences — ASM
TERMEM.C2	Modem input to disk (or other port input to disk) — ASM
M.C2	Output a file to modem (or another port) — ASM
PRINT.C3	Parallel (enhanced) printer driver — ASM
MODEM.C2	TTL output to CRT and modem (or other port) — ASM
SCIPKG.C1	Scientific math routines — PASCAL
U.C4	Mini-monitor, disk resident, many useful functions — ASM
PRINT.C4	Parallel printer driver, without PFLAG — ASM
SET.C5	Set printer modes — ASM
SETBAS1.C5	Set printer modes — A-BASIC
	(And many more)

**Over 30 TEXT files included in ASM (assembler) — PASCAL — PIC (position independent code) TSC BASIC-C, etc.

NOTE: .C1,.C2, etc. = Chapter 1, Chapter 2, etc.

This will be a limited run and we cannot guarantee that supplies will last long. Order now for early delivery.

Foreign Orders Add $4.50 S/H

Softcover — Large Format

Book only: **$7.95** + $2.50 S/H

With disk: 5" **$20.90** + $2.50 S/H

With disk: 8" **$22.90** + $2.50 S/H

Finally I don't have to type in all that source code, a GREAT, GREAT idea. Thanks and good luck, you guys are the best!
TOA

I will keep this (FLEX USER NOTES) boo always beside my computer. I have not ever, with any magazine, received as muc information as this one book, and 68 MJ.
DIB

See your local S50 dealer/bookstore or order direct from:

Computer Publishing Inc.
5900 Cassandra Smith Rd.
Hixson, TN 37343
(615) 842-4601

Telex 5106006630

™FLEX is a trademark of Technical Systems Consultants

FLEX User Notes

Ronald W. Anderson
3540 Sturbridge Court
Ann Arbor, MI 48105

More Reader Feedback

I've received a few more letters from rea ers, now some three months after my request for feedback. The one common note in virtually all of them has been the request to continue the discussions of FLEX and how it works. It has been pointed out to me that most of the FLEX users don't have the "Advanced Programmer's Guide". I do recall that I didn't get one with my original FLEX from SWTPc but shortly after receiving FLEX, I received an ad indicating that I could purchase a copy for $25. I did so immediately. The story is that SWTPc never did supply it with their version of FLEX. TSC started nearly immediately including it with their FLEX manual. Don Williams estimates that about 20% of all FLEX users have that manual. I'm going to ask Don to tell you here whether it is still available anywhere (e.g. from TSC directly or from SWTPc).

Don's Answer: the advanced programmers guide is available from S.E. MEDIA, price $25.00 + $2.55 S/H = $27.50 in all.

If it is not to be had, we will just have to start discussing the subjects covered in it here in this column.

Don's Answer #2: Let's save them the $27.50 and discuss it here. O.K., fellows (and gals)?

Several readers sent responses that were not really responses to my questions in the June column but requests for information. I have tried to answer as many of those as I could by mail. I honestly don't think there were many specific questions of general interest that I have not mentioned here. I received a letter from Kent Meyers yesterday to tell me that I gave a wrong answer in the September column to the person who asked advice on how to get OS-9 up on a SWTPc system. (I prefaced my answer with the statement that I was the wrong person to ask about OS-9). Kent tells me that Peripheral Technology has a version of OS-9 that runs on their PT-69 system (I was aware of that) and therefore will run on SWTPc hardware (I didn't make the association as I should have). Kent didn't go on and elaborate on that, but I know that the FLEX that will run on the PT-69 will also run on a SWTPc system with the DC-4 type disk controller. I suppose OS-9 would require that level disk controller also.

Editors

I've essentially finished my little editor project's first phase, that of getting it written and debugged in PL/9. I'm happy with it, it is just what I always wanted in an editor. I've done all the things I told the other suppliers of editors they ought to do to improve their product. (Only two listened to my comments and actually changed their software). Reports of the possible market among 6809 users are disappointing, but I suppose I would have done this for my one amusement anyway. I am going to send out some copies for testing and comments. You may see a review in '68' if the response is positive enough to convince Don Williams to market it through South East Media.

Don's Answer #3: S.E. MEDIA has already accepted "PAT" the Ron Anderson editor. If you ever wanted the famous "PIE" text editor. well this will sure make your day. it is a lot like it, only better! See S.E. MEDIA catalog, this issue or next.

Tandy

The following is not directly related to FLEX Users, and you may express your displeasure to me if you wish, for my taking up space here, but I think it is relevant to computing, and it gives you a point of reference with regard to how we are treated by our 68XX suppliers in general. I suppose you could call this an "open letter" to Tandy.

As most of my readers know, the company for which I work bought a Tandy 1200-HD IBM compatible computer in January of this year. We immediately had problems trying to run some third party software that had run fine on an IBM PC. After weeks of frustration with Tandy telling us that the problem must be ours, they finally admitted that there were some bugs in their monitor ROMs and in the then current version of MS-DOS. I have a letter from Dallas dated May 10 in response to my request that they allow us to test their "fix" of the bug. The letter essentially says that the patch will be available any day now, and we wouldn't get the bug fixed any sooner by becoming a test site for Tandy.

We had heard nothing further regarding the fix by the beginning of July, so we asked the local Radio Shack about the status of it. We ended up calling Dallas and were given a part number under which to order the fix. On August 1, we called the Radio Shack again to be told that the fix was not yet in but that they would look into it. After a day of phone calls again, we found that it had been available for a month, under a different number, and that Tandy has classified it as an "upgrade" and that we are supposed to pay $35 for the parts and their labor to install the ROMs.

At this point we are seeing purple. The fix is perhaps incorporated in an upgrade to their system, but the system simply doesn't work as it was supposed to from day 1 without the "upgrade". At this point I'd have to conclude that the smart thing to do would be not to buy computers from Tandy. Of course $35 is a trivial amount to pay for the cure of the problem. We spent something more than the cost of the computer on salaries for people fooling around and trying to find "our problem" in the system not working properly in the first place. The fact that Radio Shack wants to CHARGE us for the fix is just a case of adding insult to injury. Don't get me wrong, the 1200-HD is a terrific buy, and it works fine if you don't try to use the part that doesn't work. (The serial port works fine until you try to use the handshakes, then the computer dies instantly). On the promise that the problem was known and the fix was forthcoming, we bought two more of them and they have been reliable except for the above mentioned problem.

Our displeasure is therefore not with the computer but with the organization and their user support. Though they seem well equipped to han le customer service in the vein of "which software suits my needs best?" and "What cable do I buy to connect a model XYZ printer to my computer?" There seems to be no provision for handling legitimate bug reports. We have continually found that if we talk to four people in the Tandy organization we get five different stories, and that three of them have changed their story by the next day! Come on you guys! You have an almost great product at "the lowest price in town". Why can't you get it together and give your customers some service???

Don's something or another #4: Sounds like stuff I have been saying for years. Maybe you sorta understand now why we folde e, in the black, Tandy oriented magazine. I got tired of hyping an outfit that was all grin and teeth. I owed my readers more. And as a consultant guess what I tell clients. Maybe there are some others like myself who can't be brow beaten. Gosh, I also wonder why Tandy's computer market share fell so fast in the last year or so? Huh?

Don't think I've got a lock on things though. I once, years ago, bought an Edsel. Don.

Peripheral Technology

We took our daughter Pam to Holland Michigan to Hope College last Saturday. Pam had been using my old SWTPc system or the PT69 that I had set up alongside for the past couple years to do reports and term papers. When I offered to put a system together for her to take to college the idea was received more warmly than I had expected. We found a used Heath H-19 terminal that has a cutout for a disk drive. I saw the ad for the 2/3 height disk drives in '68' a few months back and ordered two for $85. I figured I could squeeze an extra 1/3 disk drive into that Heath case, and with the aid of saws and files, I managed to make them fit (just barely). I foun a power supply in the Jameco catalog for $39.95 and ordered it. A few Saturdays ago, I got down on the floor (no room on any bench area in my computer room), and hacked away. Before the day had passed, I had the drives installed via some home made brackets, and had the power supply mounted and wired.

I was a bit disappointed to see that the power supply interfered with the video a little, and more disappointed to find that the drives didn't seem to read reliably. I started playing with shielding between the power supply and the neck of the CRT, and found that a piece of aluminum foil in a paper envelope, grounded via a clip lead and placed between the supply and the CRT just about eliminated the problem with the video. I had the thought that maybe the power supply was interfering with the disk drive read heads, and put a shield over the drive as well. The read/write problems vanished. Later I found my old AC-30 cassette interface and did a little cutting and bending on the perforated cover to make a shield for the entire power supply. (I guess I am like Chuck Adams - see letter in the September issue - in that I never throw any computer equipment away either). *#5: I got two of them! AC-30s and entire 6809 computers in M19 cabinets, some of the old and new - everything except printers. Don.* With the shield in place and grounded, there is NO interference with the video, and the drives work fine as well. I connected the PT69, though I didn't put it inside the terminal case, and added an Epson MX-80 that I had been using for a while. A few nights of testing the software, and we were all set.

I expect a few phone calls with "How do I" questions, but I think the system will be quite reliable, and it all fit on a small Dormitory style desk in Pam's room. Of course I supplied my own Editor and text formatter. Classes start tomorrow, and I expect by the end of the week to receive a call or two. Fortunately Holland is only about 160 miles away. Last year about this time I helped the son of one of the '68' readers who was here at the University of Michigan to get his PT69 system going. Any readers over in the Holland area (Hudsonville, Jenison) who might be willing to help Pam in an emergency? Problems are likely to be procedural rather than hardware.

FLEX Problem

I found something rather hard to believe a few days ago. I've been adding features to my Editor, and added the ability to call a FLEX utility while editing. I found that DIR (CAT) worked just fine, but when I tried to LIST a file, though it seemed to work, I found that the editor crashed when I tried to exit and save the file I was editing. The problem was traced to the standard, supplied with FLEX, LIST utility. It seems that the author took a shortcut and called the FLEX FMSCLS routine ($D403) to close the file after listing it. According to the FLEX manual, FMSCLS is an emergency routine that closes all open files. It can be called if a program fouls up or a disk file is unreadable, or some other terrible error occurs.

In the case of my editor, I open the output file at startup, and LIST was nicely closing it for me though there was no error in the LISTing process whatever. I disassembled LIST.CMD and changed the exit to close the file normally even in case of an error. There is no harm done if the file can be closed, and if it can't using FMSCLS won't close it any "better". That way, the error message from LIST is still reported, and an error such as "FILE NOT FOUND" won't bomb the saving of the file being edited.

Perhaps you have noticed that with LIST, if you specify line numbers, you get numbers in the format 23.00= followed by the line of text. That format for line numbers is compatible with the old TSC EDIT program, but most of us have different editors now. I changed the form to the line number followed by two spaces, as in "23 ". I like it much better. I will supply the reworked LIST utility with my editor

Software

A few months ago I wrote a bit here about trying some editors for the Tandy (IBM PC Clone) system. I mentioned one particularly bad one and that I had sent a letter of complaint to the supplier and to PC World Magazine. Several months have gone by and I can report that the supplier didn't bother to respond to my letter. PC World has not acknowledged the letter to them either, let alone printed it, though the time has been considerably shorter since I wrote them about the problem.

As a matter of record, I have NEVER been ignored by a supplier of FLEX software, including Jack Hemenway way back when I bought Strubel. I wrote Hemenway Associates a rather negative letter, and Jack himself called me to discuss my criticisms. When the company bought Stylograph, Bob Bundy, the author, replied to my letter, but not only that, he made a few changes in Stylo at my suggestion. He explained that one of our problems was due to the hardware of the CT-8212 terminal we were using. I bought Lucidata Pascal way back, and I still correspond with Nigel Benee now and then. My letter to him was most enthusiastic, and was later quoted in their ad in '68'. The company bought OmegaSoft Pascal, and we reported a few bugs. These were all cured very quickly, and we continued correspondence with Bob Reimiller at OmegaSoft for some time. I mention these in particular, because they all happened BEFORE I began to write for '68', so I was "just another customer" at the time.

A little later, when Windrush started selling PL/9 I was sent a copy for review. I now correspond with author Graham Trott and Bill Dickinson at Windrush in England. I have also spoken to James McCosh, author of the "C" compiler FLEX version sold by Windrush and OS-9 version sold by Microware. I have had some contact with virtually every supplier of 6809 software (even Bruce Dugger of Dugger's Growing Systems, whose software was a disaster). *Gosh! - #6: Ron, it got better, sorry he didn't have it right at first, we needed a good C then, got a couple good 'uns now. Don.* I'd like to think that my input encouraged the suppliers and authors to improve their software. I'm not used to being ignored, and I don't like it at all. We 6809 users ARE lucky!!

IS FLEX ROMMABLE?

One of the response letters I received recently mentioned that the writer was thinking about a ROM version of FLEX. I wrote back that it would be a very large project. FLEX is so full of locations that must be RAM, that some clever approach would have to be taken to make it compatible with the standard version. Address $C000 to $C07F are used for the system stack when FLEX is running. C080 to C0FF are the input buffer used to store the command line that you typed in. $C100 to $C6FF is RAM into which most FLEX Utilities are read and from which they are executed. This area is called the "Utility Command Space". $C700 to $C83F is supposed to be a file control block connected with Print Spooling, but I understand that some SWTPc versions of FLEX have some code in this area to make print spooling work.

To continue, $C840 to $C97F is the "system FCB" area, the place used by FLEX as a file control block to load the Utilities. $C980 to $CBFF are two file control blocks used in conjunction with the I.CMD and the O.CMD. We're now through 3K of FLEX and there is almost no code yet. Now the FLEX system parameters (TTYSET values, Pause, System Drive, Working drive, System DATE registers, etc.) are located from $CC00 to $CCBF. The Printer Drivers are located from $CCC0 to $CCFF. So far, all of this has to be RAM. Starting at $CD00 through $CD4E are jump tables into FLEX. Various versions of FLEX have the different routines at different locations, but these JUMP VECTORS are always in the same place. Some of these vectors are MODIFIED by FLEX as it runs. For example the OUTCH jump at $C00F is modified when you output to a printer. It therefore must be RAM also. The code from $CD4E through $D3FF is the code for all the routines in the jump table, such as RPTERR the error reporting routine, PCRLF, to print a CR and LF, GETFIL to get a file name from the command line, INDEC and OUTDEC to get a decimal number in from the command line and print it out respectively, and several others. $D400 TO $DDFF is the File Management System code, and $DE00 to $DFFF is the disk hardware driver code space.

I think my point is made that a very large part of FLEX MUST be in RAM. Since we can't intersperse RAM and ROM on an address by address basis, there is little of FLEX that would be rommable. One possibility that I can think of would be to put FLEX in ROM at some high address (extended address, using the DAT, that is), and fix the start vector in the monitor ROM to call a routine to download FLEX from ROM to its normal location in RAM, and jump to the coldstart location. That would be faster than booting it from a disk, takes no extra memory (unless you had a full megabyte of RAM implemented via extended addressing), and would relieve the system disk from having to have FLEX on it.

You might wonder why FLEX was written in this way so that it is so hard to put in ROM. When FLEX was written, the EPROM was not yet a practical reality. the Monitor Roms were mask programmed, a process that has a very large set-up cost and a reasonable per unit cost. No one, I'm afraid, gave second thought to the possibility of someday putting FLEX in a ROM. If they had thought about the possibility, it would have been fairly easy to write it so all the variable parts of FLEX were in one contiguous block and all the fixed parts were in another contiguous block. I suspect it would have turned out approximately so that $C000 to $CFFF would be RAM and $D000 to $DFFF would be ROM. There would have to be some additional ROM somewhere with routines to initialize the RAM to nominal values, which is now done by loading data into the variable area. Of course one could write an operating system similar to FLEX now, with variables and program separated so that the program part would be rommable, but it couldn't be made compatible with FLEX or any of the software that runs under FLEX, so it would not be a very fruitful task.

Well, I've used a question about FLEX as a ruse to give you a quick memory map of the various parts of FLEX. Perhaps one of these times we will get into some detail about the various "USER-CALLABLE SYSTEM ROUTINES".

- - -

"C" User Notes

Edgar M. (Bud) Pass, Ph.D.
1454 Latta Lane
Conyers, Ga 30207

This chapter discusses the use of random file access functions in the C language, with particular attention to the use of the McCosh C random file access functions under FLEX. It illustrates the use of these functions with an example program.

RANDOM FILE ACCESS IN C

Since random file access is somewhat more complex than sequential file access, there is a reluctance of many programmers to use it in many applications in which it should be used. This section discusses random file access and attempts to remove some of the mystery surrounding it.

Random file access is normally provided in C language implementations with four functions: "fseek", "ftell", "lseek", and "rewind". All except "lseek" work with file pointers; "lseek" works with lower-level file descriptors.

Each function is described separately, then their combined usage is discussed. Note that the comments about the usage of the functions under FLEX is restricted to the McCosh C compiler, as it is the only implementation which currently supports random file access under FLEX.

```
     fseek(fileptr, offset, place)
     FILE *fileptr;
     long offset;
     int place;
```

The "fseek" function attempts to reposition a file such that the next character read from or written to a file is at a specified offset. The second parameter is taken as an offset from the following points:

place	point
0	beginning
1	current
2	end (not for FLEX)

The offset from the end of the file is not allowed in FLEX because the lack of a byte count for the last sector in the FLEX disk format implies an uncertainty of exactly where the end of a FLEX disk file really is. The use of "fseek" is not allowed on a sequential FLEX file open for writing since FLEX does not support the extension of an existing sequential FLEX file.

The McCosh implementation of "fseek" for sequential FLEX files is somewhat inefficient, as the FLEX file structure does not support the random addressing of sequential files. The implementation for random FLEX files is not as inefficient, as FLEX supports random access, although somewhat crudely, in this case. For locations forward in a sequential file from the current location, "fseek" reads each byte until the byte before the desired one has been read. For locations backward in the file, "fseek" first rewinds the file before reading the bytes to be skipped. Even though the McCosh manual indicates that "fseek" works with buffered files, it currently works only with unbuffered ones.

```
     long ftell(fileptr)
     FILE *fileptr;
```

The "ftell" function returns the current offset relative to the beginning of the file. This is normally the number of bytes required to be skipped to reach the current byte position from the beginning of the file.

```
     rewind(fileptr)
     FILE *fileptr;
```

The "rewind" function repositions a file at the beginning. It is equivalent to "fseek(fileptr,(long)0,0)".

```
     long lseek(filedesc, offset, place)
     int filedesc;
     long offset;
     int place;
```

The "lseek" function is used identically to the "fseek" function except that it is used with low-level I/O and that it returns the current offset into the file.

To obtain the equivalent of the "ftell" function with low-level I/O, "lseek" may be called as "lseek(filedesc,(long)0,1)".

To obtain the equivalent of the "rewind" function, "lseek" may be called as "lseek(filedesc,(long)0,0)".

The following program demonstrates several of the functions just described. It may be applied to any text file with at least about ten records.

```
/* demo program for random file access functions */

#include <stdio.h>
#include <ctype.h>
main()
{
        char buf[256];
        FILE *fp;
        long lx, ftell();

        pflinit();        /* required for McCosh C */
        printf("tellseek test\n");
        if ((fp = fopen(argv[1], "r")) == NULL)
           exit(1);
        setbuf(fp,NULL); /* required for McCosh C */
        printf("tellseek file open\n");
        printf("%s",fgets(buf, 250, fp));
        printf("%s",fgets(buf, 250, fp));
        printf("tellseek ftell = %08lx\n",
                lx = ftell(fp));
        printf("%s",fgets(buf, 250, fp));
        printf("%s",fgets(buf, 250, fp));
        printf("tellseek fseek %08lx = %04x\n",
                lx, fseek(fp, lx, 0));
        printf("%s",fgets(buf, 250, fp));
        printf("%s",fgets(buf, 250, fp));
        printf("tellseek ftell = %08lx\n",
               ftell(fp));
        printf("tellseek fseek %08lx = %04x\n",
               lx, fseek(fp, lx, 0));
        printf("tellseek ftell = %08lx\n",
               ftell(fp));
        printf("%s",fgets(buf, 250, fp));
        printf("%s",fgets(buf, 250, fp));
        printf("tellseek rewind\n");
        rewind(fp);
        printf("tellseek ftell = %08lx\n",
               ftell(fp));
        printf("%s",fgets(buf, 250, fp));
        printf("%s",fgets(buf, 250, fp));
        exit(0);
}
```

In some applications, the overhead associated with the FLEX McCosh C random file access functions, when applied to sequential files, may be prohibitive. The use of FLEX random files may not be appropriate in many applications, so an alternative which avoids both would be useful.

The replacement functions provided below for "ftell", "fseek", and "rewind" avoid the overhead associated with the corresponding standard functions, but are more limited in usage. They allow the direct access to any byte in a file without passing over intermediate bytes, which is much faster than the alternative. They may be also be used in versions of McCosh C for FLEX which do support the random file access functions described earlier. They may be used in Introl C, if you can successfully add the equivalent of the "flexfms" function to the Introl C library.

The primary restriction is that the "offset" parameter for "fseek" is a long integer, but is not a byte count, and must be the result of "ftell". The "place" parameter must be zero, indicating positioning from the top of the file. This allows the direct positioning of a file to a specific byte, but the file must be either read or prepared first. The long integer is actually composed of the drive number, track number within drive, sector number within track, and byte number within sector, providing an absolute, not relative, description of the byte position.

An alternative to this technique would be to require that the file be allocated contiguously and sequentially and without compression. Then the address of a byte could be computed directly, knowing the number of sectors per track. This is exactly the technique many mini-computer operating systems use to support a form of random access to otherwise sequential files. Of course, there are many other techniques which are used to support true byte-addressable file structures.

```
/* the following include and functions replace
   functions of corresponding names provided
   in the FLEX McCosh C Standard I/O Library */

#include <flex.h>

long ftell(fp)
FILE *fp;
{
        char *p, *q;
        long lx;

        p = get_fcb(fp);
        q = &lx;
        *q = *(p + 0x03);
        *(q + 1) = *(p + 0x1e);
        *(q + 2) = *(p + 0x1f);
        *(q + 3) = *(p + 0x22);
        return lx;
}

fseek(fp, offset, place)
FILE *fp;
long offset; /* must be from ftell or zero */
int place;   /* must be zero */
{
        char *p, *q;
        int res = 0;

        if (place)
            return (-1);
        p = get_fcb(fp);
        q = &offset;
        if ((*(p + 0x03) != *q) ||
            (*(p + 0x1e) != *(q + 1)) ||
            (*(p + 0x1f) != *(q + 2)))
        {
            if (*(p + 0x02) == 0x02)
                res = flexfms(p, 0x0a);
            if (*(p + 0x1f))
            {
                *(p + 0x03) = *q;
                *(p + 0x1e) = *(q + 1);
                *(p + 0x1f) = *(q + 2);
            }
            else
            {

              *(p + 0x1e) = *(p + 0x11);
              *(p + 0x1f) = *(p + 0x12);
              *(q + 3) = 0x00;
            }
            res |= flexfms(p, 0x09);
        }
        *(p + 0x22) = *(q + 3);
        *(p + 0x3b) &= 0x80;
        return res;
}

rewind(fp)
FILE *fp;
{
        return fseek(fp,(long)0,0);
}

flexfms(fcb, fcn)
int *fcb, fcn;
{
#asm
 ldx $06,s   get fcb pointer
 lda $09,s   get function code
 ldb ,x      remember function code
 sta ,x      set function in fcb
 clr $08,s   zero return code
 clr $09,s
 pshs b,x
 jsr $d406   call flex fms
 puls b,x
 beq f_l_x_  check for error
 dec $08,s   mark error
 dec $09,s
f_l_x_
 stb ,x      restore function code
#endasm
        return fcn;
}
```

This is the beginning, not the end, of the information on the use of random file access. More complex file structures, such as index files, hashed files, index-sequential files, B-tree files, etc. may be defined using these basic functions. The C problem will ask the reader to define a very simple self-indexed file structure. Other file structures will be discussed in the future.

C PROBLEM

The previous C problem was the extension of the string substitution program to become a variable substitution program. The program provided below accomplishes this, accepting a file containing a list of substitutions; it might be used in making the variable names used in a C program more meaningful. Its invocation sequence is as follows:

```
     variable oldfile wordlist newfile
```

```
/* copy file with variable substitution. */

#include <stdio.h>
#include <ctype.h>

#define TABSIZE 300
#define ENTSIZE 40
#define LINESIZE 256

char comment = 0, cprog = 1, table[TABSIZE][ENTSIZE+1];
int tabsize = 0, tabsize2 = 1;

main(argc,argv)
int argc;
char *argv[];

    char line[LINESIZE+1], line1[LINESIZE*2+1], *x, *y;
    char strcmp();
    FILE *input, *output, *word;
    int count = 0, c = 0, p = 0, z;

    putc ('\n', stderr);
    if (argc < 4)
    {
            fputs ("variable substitutes variable names\n", stderr);
            fputs ("use: variable oldfile wordlist newfile [-n]\n",
                stderr);
            fputs (" wordlist records look as follows:\n", stderr);
            fputs ("  old-var <space> new-var\n", stderr);
            fputs (" -n option for non-c programs\n", stderr);
            exit (1);
    }
    if (((input = fopen (*++argv, "r")) == NULL) ||
        ((word = fopen (*++argv, "r")) == NULL) ||
        ((output = fopen (*++argv, "w")) == NULL))
    {
        fputs ("can't open file ", stderr);
        fputs (*argv, stderr);
        exit (1);
    }
    if (argc > 4)

        --cprog;
    fputs (" reading wordlist\n", stderr);
    while (fgets(x = y = table[tabsize], ENTSIZE, word) != NULL)
    {
        if (*x)
            if (++tabsize >= TABSIZE)
            {
                --tabsize;
                if (!p)
                {
                    fputs (" too many entries\n ", stderr);
                    p = 1;
                }
            }
            else
            {
                while (*x++ > 0x20);
                if ((*--x < 0x20) || (*(x + 1) < 0x20))
                {
                    fputs (" invalid entry:\n ", stderr);
                    fputs (y, stderr);
                    p = 1;
                }
                else
                    *x = '\0';
            }
    }
    fclose (word);
    if (!p)
    {
        qsort (table, tabsize, ENTSIZE+1, strcmp);
        for (c = 1; (c <= tabsize); ++c)
        {
            if ((!strcmp (x = table[c], y = table[c - 1])) &&
                (!strcmp (x + strlen (x) + 1, y + strlen (y) + 1)))
            {
                fputs (" duplicate old-var at\n ", stderr);
                fputs (x, stderr);
                putc (' ', stderr);
                fputs ((x + strlen (x) + 1), stderr);
                ++p;
            }
        }
    }
    if (p)
    {
        fputs (" terminated due to word list errors\n", stderr);
        exit (p);
    }
    fputs (" copying old file to new file, with changes\n ", stderr);
    while ((tabsize2 <<= 1) < tabsize);
    tabsize2 >>= 1;
    while (fgets(line, LINESIZE, input) != NULL)
    {
        count += strchange (line1, line);
        fputs (line1, output);
    }
    fclose (input);
    fclose (output);
    for (p = 0, z = 10000; z; z /= 10)
    {
        if ((c = count / z) || p || (z == 1))
        {
            putc ((c + '0'), stderr);
            count -= c * z;
            ++p;
        }
    }
    fputs (" variables changed\n", stderr);
    exit (0);
}

/*
    Bsearch (name) finds an item in the variable table by binary search.
    Name points to the variable to be found, null terminated.
    It returns a pointer to the new variable or NULL.
*/
char *bsearch (name)
char *name;
{
    char *s;
    int c, t, i = 0, p = tabsize2;

    if (t = strcmp (name, (s = table[p - 1])))
    {
        if (t > 0)
            i += (tabsize - tabsize2);
        do
        {
            if ((t = strcmp (name, (s = table[c = i + (p >>= 1)]))) > 0)
                i = c;
        }
        while (t && p);
    }
    return (t ? NULL : (s + strlen(s) + 1));
}

/*
    Strchange(dst, src) copies the input string to the output string.
    It replaces all identifiers specified in the table.
    It returns the number of replacements performed.
*/

strchange(dst, src)
char *dst, *src;
{
    char *p, *t, *v, s[LINESIZE+1];
    int c = 0, x;

    if (comment)
        goto comms;
    while ((x = *src))
    {
        if ((isalpha(x)) || (x == '_'))
        {
            for (t = s, p = src;
                (isalpha(*src) || isdigit(*src) || (*src == '_'));
                *t++ = *src++);
            *t = '\0';
            if ((v = bsearch (s)) != NULL)
            {
                while (*dst++ = *v++);
                dst -= 2;
                ++c;
            }
            else
            {
                while (p != src)
                    *dst++ = *p++;
            }
        }
        else
        if ((x == '\'') || (x == '"'))
        {
            do
            {
                if (cprog)
                {
                    if (*src == '\\')
                    {
                        *dst++ = *src++;
                        if (!*src)
                            goto exit;
                    }
                }
dquote:
                *dst++ = *src++;
            }
            while (*src && (*src != x));
            if (*src >= 0x20)
            {
                *dst++ = *src++;
                if ((!cprog) && (*src == x))
                    goto dquote;
            }
        }
        if (cprog && (x == '/'))
        {
            *dst++ = *src++;
            if (*src != '*')
                continue;
            ++comment;
            do
            {
                *dst++ = *src++;
comma:

                ;
            }
            while (*src && (*src != '*') && (*(src + 1) != '/'));
            if (!*src)
                break;
            *dst++ = *src++;
            *dst++ = *src++;
            --comment;
        }
        else
            *dst++ = *src++;
    }
exit:
    *dst = '\0';
    return (c);
}
```

The next C problem involves the construction of a C program (using the random file access functions described earlier) which will provide a fixed response to a keyword or phrase input to it. One application of this program would be the provision of a "help" facility for a system.

The input file is assumed formatted as follows:

```
*keyword or phrase
response first line
:
:
response last line
*keyword or phrase
response first line
:
:
response last line
:
```

This file format will produce a self-indexed file structure. In use, the program would be executed and would accept one line of input at a time, providing the fixed response from the file, if there is one, and would terminate on a null input line.

EXAMPLE C PROGRAM

Following is this month's example C program; it lists several files in parallel vertical columns, perhaps to use in comparing similar data or similar files manually. Note that McCosh C and most small C compilers do not support the continued strings used in the HELPMSGx definitions. However, they may readily be split into multiple shorter strings in order to make them acceptable to those compilers.

```
/* program to list files in parallel vertical columns
     by Edward K. Dunham */

#include  <stdio.h>

#define   MAXFILE   23   /* max number of input files */

#define   HELPMSG1 \
  "\7Usage: LONGSIDE [<file> [<width>] [<spaces>]].\n"

#define   HELPMSG2  "If <width> is null or zero, the whole \
input line is sent to output.\nFor <spaces> to be recognized, \
<width> must be given explicitly, even if zero.\n"

#define   HELPMSG3  "The first numeric argument after an \
alphabetic is taken as <width>, the last\nas <spaces>.  There \
is no way to specify leading spaces.\n"

char argnr,                 /* argument index */
  done,                     /* flag signalling completion */
  filenr,                   /* input file index */
  line[MAXLINE],            /* input buffer */
  nfiles,                   /* input file count */
  *s,                       /* character pointer */
  spaces[MAXFILE],          /* spaces to be output after line */
  w,                        /* width (temporary) */
  wflg[MAXFILE],            /* flag signalling width limit */
  width[MAXFILE];           /* width */
FILE *infp[MAXFILE],        /* file pointer */
  *fopen();

main(argc, argv)
int argc;
char *argv[];
{

/* give instructions if call lacks arguments */
  if (argc < 2) {
    fprintf(STDERR, HELPMSG1); fprintf(STDERR, HELPMSG2);
    fprintf(STDERR, HELPMSG3);
    exit();
  }

/* process command line and open input files */
  for (argnr = 1; argnr < argc && *argv[argnr]; ++argnr) {
    if (isalpha(*argv[argnr])) {
                  /* an alpha argument can be only a filename */
      ++filenr;
      if ((infp[filenr] = fopen(argv[argnr], "r")) == ERROR) {
        fprintf(STDERR, "\7Can't open %s.\n", argv[argnr]);
        exit();
      }
      wflg[filenr] = NO;          /* start with no width limit */
    }
          /* if arg is numeric and filename but no width is set,
                                                arg must be width */
    else if (isdigit(*argv[argnr]) && infp[filenr])
      if (!wflg[filenr]) {
        width[filenr] = atoi(argv[argnr]);
        wflg[filenr] = YES;
      }
             /* if width set, any other numeric arg is spaces */
      else spaces[filenr] = atoi(argv[argnr]);
                            /* no other kind of argument allowed */
    else {
      fprintf(STDERR, "\7Wrong syntax.\n%s", HELPMSG1);
      exit();
    }
  }
  nfiles = filenr;

/* for each line, read a line from each input file in turn */
  do {                         /* continue until all files finished */
    done = YES;                  /* start by assuming we're done */
    for (filenr = 1; filenr <= nfiles; ++filenr) {
      w = width[filenr] ? width[filenr] : 1;
      if (s = fgets(line, MAXLINE - 1, infp[filenr])) {
        done = NO;                 /* not done if any input left  */
        while (w && *s && *s != '\n') {
          putchar(*s++);         /* fill to width or end of line */
          if (width[filenr]) w--;   /* count only if width set */
        }
      }
             /* now pad with spaces and add separating spaces */
      if (width[filenr]) while (w--) putchar(' ');


      for (w = spaces[filenr]; w; w--) putchar(' ');
    }
    if (!done) putchar('\n');  /* newline after all files read */
  } while (!done);

}
```

OS-9 User Notes

Peter Dibble
19 Fountain Street
Rochester, NY 14620

In Praise of Assembly Language

Don't get me wrong, I'm all in favor of compiled languages. I just think that there is still a role for assembly language. If Ron Anderson will forgive me for a moment, I feel the need to get my feelings on this subject on record.

I don't write in assembler very much any more, but I've written more thousands of lines of 370, 8080, Dec-20, and 6809 assembly language than I care to remember. I've even done some coding in raw machine language for a Univac. The compiler fanatics point out that human time is more expensive than machine time and that compilers produce much better code than they used to. They don't notice that some programs are not throw-aways. Consider the system sort program. The world's computers are said to spend about half their time sorting. If you write a sort that runs a little faster than the competition, you will be surrounded by DP people waving money at you.

IBM writes a lot of stuff in PL/S (a compiler). They might have written some of their sort programs in it. Two points: PL/S is very like assembly language; IBM sells hardware.

The claim that I sometimes hear that a given compiler produces better code than assembler strikes me as silly. The Forth contingent sometimes make the related claim that a program in Forth requires less memory than a similar program in assembler (I think they are talking about storage for the executable form, not the program source). If you get into this argument try this thought experiment: ask the Forth or compiler promoter to code up their favorite demonstration program; test it and make appreciative noises; run it through a good disassembler; assemble the result. You now have an assembly language program exactly as good as the compiled (or threaded) program. Compilers don't do anything magic.

The best compilers spend much time playing with register allocation and shuffling code around to get good performance. A good assembly language programmer does the same thing, but a compiler can do the rote part of the work much faster. It may even be able to do some optimizations that the human mind couldn't handle. Still, I don't think anyone would put a compiled program up against a hand-coded assembly language program if programmer time were no object.

Microware evidently feels that the OS-9/68K operating system gets enough use to be worth hand coding, but the utilities are not. I hope this is a temporary thing. I'd like to believe that there is a team at Microware busily rewriting utility programs in assembler, but I don't. OS-9 is known as a small, fast operating system; justifiably -- it is. The 68K kernal lives up to this reputation. The utility programs do not.

If Microware had written the entire OS-9/68K system in assembly language it would probably barely be in beta test. The utility programs definitely wouldn't have all the nifty features they do. I'm glad it's out and I'm glad the features are there. Now that the software is in use they can discover that copy, and dir, and rename, and del, and acred are heavily used. Those programs can be recoded into assembler using the existing C programs as guidelines.

The compiler fanatics point out that human time is more expensive than machine time and that compilers produce much better code than they used to. They don't notice that some programs are not throw-aways.

Let's say that Microware decided to recode just one program, the C compiler. They might be able to double its speed. They should be able to shrink it some too. Would it be worth it? Let's say it takes them four man-years to recode it. If 200 people were waiting a total of a half hour each day for C compiles to complete they could save a total of fifty hours per day just by using the faster compiler. At that rate it would take about eight months for the faster compiler to pay for itself.

The case for recoding the C compiler isn't as good as I make it sound. Assembly language programs are harder to maintain than high-level language programs. They are also much harder to move from one type of computer to another. The time it takes to write a program in assembly language is just the first part of the cost. The additional difficulty of maintaining assembly code is probably a more important cost than the difficulty of writing it.

Nice slow compilers encourage people to do something useful while they run. It could be that the C compiler is slow enough to be used that way; doubling the speed could change people's work habits and actually cost time. I don't think this would happen with Microware C. My compiles only take more than about a minute when I recompile an entire system (which I do rarely).

Efficiency for the user demands that programs be carefully written in assembly language. Efficiency for the vendor dictates compiled languages. There is a compromise. I have used it with a great deal of success. First you write the program in C. You debug it and improve the algorithm as much as possible. When it is finely polished, analyse it or use the C profiler (preferably both). Almost every program will have some parts that are much more heavily used than others; make those parts into separate functions and put them in a separate file. Run through the debugging and polishing routine again. Get the C compiler to generate unoptimized assembly language with the C source lines included as comments. Hand optimize the assembly code using all the clever tricks you like. Assemble it and link the whole program and you're done.

The program resulting from this kind of selective hand-optimization will be practically as fast as an equivalent program that was written entirely in assembler and it is likely to be a better program in other respects. It won't be nearly as small as the equivalent assembler program, but perhaps that isn't as much of an issue as it once was. I am especially pleased to be able to sharpen and debug my algorithms in C. The compiler also lets me concentrate more on the less used parts of the program.

If I were writing everything in assembler, fancy error recovery and other amenities would take a lot of time and might not get done very carefully. That kind of code is seldom used. The speed of the program isn't damaged noticeably when sections of code that are only used a few times are written in plain C.

A painless way to include highly optimized code in C programs is to write the libraries, or at least the most CPU intensive parts of them, in assembler. Microware hasn't done that, but there is nothing preventing me from making a start at it. The string-handling functions are good examples of code that often is very heavily used.

Microware's string handling functions can be replaced in at least two ways. The individual ".r" files containing the string functions can be given on the command line of each compile of a program that uses them, or the ".r" files can be appended at the beginning of the C library file. The second solution is easier to use, but it doesn't work very well.

The linker, rlink, reads through the library file looking for PSects that satisfy unresolved references. When it finds them it adds them to the module it is building. If the new PSect includes external references that aren't resolved when it is merged with the new module those references are added to the list that rlink is trying to resolve.

If you put improved string functions at the beginning of the library file, they will be included in your module if they are used in your code. If they aren't, but they are used by some standard function (like printf), rlink will use the next set of string functions that it finds in the library. Microware organized the library carefully so the functions required by each function come after it in the library. The string functions belong near the end of the library, but if you put replacement string functions at the end, rlink will always use the originals (which will be before your functions in the library file). I believe that one of the programs in the Users Group library can be used to play with the C library file. It could solve these problems by replacing the string functions instead of adding new versions.

These string functions follow the descriptions in "A C Reference Manual" by Harbison and Steele. Strncat copies at most n characters but always appends a '0'; it may therefore use up to n+1 bytes. I don't think Microware C does this.

One more thing

I must have received three or four letters over the past year asking about building a new boot file with fewer modules in it than the standard boot. The problem is that os9gen takes a list of files and combines their contents into a bootstrap file. How do you get the modules in the boot file into separate files so you can select the ones you want in a new boot? Answer, the save command. The modules in the boot will all be in memory. Save them into files in whatever combinations you like.

Now a start on the string functions:

```
Microware OS-9 RMA - V1.0  85/10/18  10:49   Strings.a              Page    1
strings_a - strcat

                         psect strings_a,0,0,0,0,0
0000                     nam   strings_a
0000                     ttl   strcat
0000              strcat:
0000 ae62                ldx   2,S        get a pointer to s1
0002 3440                pshs  U
     *
     * Find the end of s1
     *
0004              catlp1
0004 6d80                tst   ,X+
0006 26fc                bne   catlp1
0008 301f                leax  -1,X       back up to '\0'

     *
     * copy s2 to the pointer
     *
000a ee66                ldu   6,S
000c              catlp2
000c a6c0                lda   ,U+
000e a780                sta   ,X+
0010 26fa                bne   catlp2
0012 ec64                ldd   4,S
0014 35c0                puls  u,pc

0016                     ttl   strncat


Microware OS-9 RMA - V1.0  85/10/18  10:49   Strings.a              Page    2
strings_a - strncat

0016              strncat:
0016 ae62                ldx   2,S        get a pointer to s1
0018 3460                pshs  U,Y
     *
     * Find the end of s1
     *
001a              ncatlp1
001a 6d80                tst   ,X+
001c 26fc                bne   ncatlp1
001e 301f                leax  -1,X       point at '\0'

     *
     * copy s2 to the pointer
     *
0020 ee68                ldu   8,S        s2
0022 10ae6a              ldy   10,S       n
0025              ncatlp2
0025 a6c0                lda   ,U+
0027 a780                sta   ,X+
0029 2706                beq   ncatlp2x
002b 313f                leay  -1,Y       check length bound
002d 26f6                bne   ncatlp2
002f 6f84                clr   ,X         terminate string
0031              ncatlp2x
0031 ec66                ldd   6,S        get S1
0033 35e0                puls  U,Y,PC

0035                     ttl   strcmp
```

```
Microware OS-9 RMA - V1.0  85/10/18  10:49   Strings.a            Page     3
strings_a - strcmp

00057 0035            strcmp:
00058 0035 ae62                 ldx   2,S        s1
00059 0037 3440                 pshs  u
00060 0039 ee66                 ldu   6,S        s2
00061
00062 003b            strcmplp
00063 003b e680                 ldb   ,X+        get byte from s1
00064 003d 2707                 beq   strcmpend  if it's zero stop
00065 003f e0c0                 subb  ,U+        (*s1 - *s2)
00066 0041 27f8                 beq   strcmplp
00067 0043            strcmpx
00068 0043 1d                   sex              change byte to integer
00069 0044 35c0                 puls  U,PC
00070 0046            strcmpend
00071 0046 e6c4                 ldb   ,U         if both strings end, return 0
00072 0048 27f9                 beq   strcmpx
00073 004a c6ff                 ldb   #-1        if only s1 ended return -1
00074 004c 20f5                 bra   strcmpx
00075
00076 004e                      ttl   strncmp

Microware OS-9 RMA - V1.0  85/10/18  10:49   Strings.a            Page     4
strings_a - strncmp

00078 004e            strncmp:
00079 004e ae62                 ldx   2,S        s1
00080 0050 3460                 pshs  U,Y
00081 0052 ee68                 ldu   8,S        s2
00082 0054 10ae6a               ldy   10,S       n
00083
00084
00085 0057            strncmplp
00086 0057 e680                 ldb   ,X+        s1
00087 0059 270c                 beq   strncmpend
00088 005b e0c0                 subb  ,U+        (*s1 - *s2)
00089 005d 2605                 bne   strncmpx
00090 005f 313f                 leay  -1,Y       check count
00091 0061 26f4                 bne   strncmplp
00092 0063 5f                   clrb             return 0 on count ending
00093 0064            strncmpx
00094 0064 1d                   sex              byte to int
00095 0065 35e0                 puls  U,Y,PC
00096 0067            strncmpend
00097 0067 e6c4                 ldb   ,U         did s2 end when s1 did?
00098 0069 27f9                 beq   strncmpx   yes, return 0
00099 006b c6ff                 ldb   #-1        no, return -1
00100 006d 20f5                 bra   strncmpx
00101
00102 006f                      ttl   strcpy

Microware OS-9 RMA - V1.0  85/10/18  10:49   Strings.a            Page     5
strings_a - strcpy

00104 006f            strcpy:
00105 006f ae62                 ldx   2,S        s1
00106 0071 3440                 pshs  u
00107 0073 ee66                 ldu   6,S        s2
00108
00109 *
00110 *
00111 *
00112 0075            strclp
00113 0075 a6c0                 lda   ,U+        *s2
00114 0077 a780                 sta   ,X+        *s1
00115 0079 26fa                 bne   strclp
00116
00117 007b ec64                 ldd   4,S        s1
00118 007d 35c0                 puls  U,PC       return
00119 007f                      ttl   strncpy

Microware OS-9 RMA - V1.0  85/10/18  10:49   Strings.a            Page     6
strings_a - strncpy

00121 007f            strncpy:
00122 007f ae62                 ldx   2,S        s1
00123 0081 3460                 pshs  U,Y
00124 0083 ee68                 ldu   8,S        s2
```

```
00125 0085 10ae6a               ldy   10,S       n
00126
00127 0088            strnclp
00128 0088 a6c0                 lda   ,U+        *s2
00129 008a a780                 sta   ,X+        *s1
00130 008c 2708                 beq   strnclr    if string ends before n
00131 008e 313f                 leay  -1,Y       check count
00132 0090 26f6                 bne   strnclp    not counted out, loop again
00133 0092            strncx
00134 0092 ec66                 ldd   6,S        s1
00135 0094 35e0                 puls  U,Y,PC     return
00136 0096            strnclr
00137 0096 313f                 leay  -1,Y       append '\0's to n
00138 0098 27f8                 beq   strncx
00139 009a 6f80                 clr   ,X+        null out
00140 009c 20f4                 bra   strncx
00141 009e                      ttl   strlen

Microware OS-9 RMA - V1.0  85/10/18  10:49   Strings.a            Page     7
strings_a - strlen

00143 009e            strlen:
00144 009e ae62                 ldx   2,S        s
00145 00a0 4f                   clra             initialize length
00146 00a1 5f                   clrb
00147 00a2            strllp
00148 00a2 c30001               addd  #1         ++length
00149 00a5 6d80                 tst   ,X+        check for end of string
00150 00a7 26f9                 bne   strllp     not end, loop
00151 00a9 830001               subd  #1         correct length
00152 00ac 39                   rts
00153 00ad                      ttl   index

Microware OS-9 RMA - V1.0  85/10/18  10:49   Strings.a            Page     8
strings_a - index

00155 00ad            index:
00156 00ad ae62                 ldx   2,S        s
00157 00af a665                 lda   5,S        ch
00158
00159 00b1            indloop
00160 00b1 6d84                 tst   ,X         check for end of string
00161 00b3 2709                 beq   indnotf    eos: not found
00162 00b5 a180                 cmpa  ,X+        check for ch
00163 00b7 26f8                 bne   indloop    not ch: check next character
00164 00b9 301f                 leax  -1,X       correct pointer
00165 00bb 1f10                 tfr   X,D        move pointer to return register
00166 00bd 39                   rts
00167 00be            indnotf
00168 00be 4f                   clra             return null
00169 00bf 5f                   clrb
00170 00c0 39                   rts
00171 00c1                      ttl   rindex

Microware OS-9 RMA - V1.0  85/10/18  10:50   Strings.a            Page     9
strings_a - rindex

00173 00c1            rindex:
00174 00c1 ae62                 ldx   2,S        s
00175 00c3 a665                 lda   5,S        ch
00176
00177 00c5            rindl1
00178 00c5 6d80                 tst   ,X+        run out to end of s
00179 00c7 26fc                 bne   rindl1
00180
00181 00c9 301f                 leax  -1,X       back up to '\0'
00182 00cb            rindloop
00183 00cb a182                 cmpa  ,-X        check for ch (backward through string)
00184 00cd 2707                 beq   rindx      found ch: return
00185 00cf ac62                 cmpx  2,S        reached beginning of string?
00186 00d1 26f8                 bne   rindloop   no: loop again
00187 00d3 4f                   clra             ch not found: return null
00188 00d4 5f                   clrb
00189 00d5 39                   rts
00190 00d6            rindx
00191 00d6 1f10                 tfr   X,D        move pointer to return register
00192 00d8 39                   rts
00193
00194 00d9                      endsect
```

Basic OS-9

Ron Voigts

TYING UP LOOSE ENDS ON STRINGS

Last time when I presented the program of the month, I mentioned briefly the form that strings can take in the OS-9 system and in C language. I feel somewhat guilty about not delving into the subject further and explaining more about it. The manipulation, storage and usage of string data is an important aspect of programming. It also can be a fun part of it.

Anything within the computer involves numbers or characters. Ultimately characters are numbers. By agreement, we all use the same code to represent characters. Actually you and I may not have agreed to it, but somebody did. And they called it ASCII code, which is short for American Standard Code for Information Interchange. ASCII is not the only code available. Some of the others are ANSI, EBCDIC and CDC Scientific.

If you aren't familiar with ASCII, check any good book on programming. Surprisingly most books touch on the character code in some manner or another. Most systems use ASCII. I won't go into great details about it, but highlight some of its finer points. The code uses 7 bits of information, allowing for numbers $00 to $7F. The first 32 characters, $00 to $1F, are control characters. They include things like carriage return, line feed, and back space. $31 to $39 are the numbers 1 to 9. $41 to $5f are the uppercase letters and $61 to $7f are the lowercase letters. The other characters are punctuations, with exception of $7f which is delete and belongs with the first 32 .

Take a bunch of characters, hang them together and you've got a string. If you've done any assembly language programming you know you can use the pseudo opcode FCC to create a character string. The line:

```
     FCC "HELLO"
```

causes the assembler to generate the ASCII code:

```
     48 45 4C 4C 4F
```

These five numbers get assembled into the object code. The OS-9 system uses another pseudo-opcode called, FCS (Force Constant String). It is entered the same way:

```
     FCS "HELLO"
```

and generates the following:

```
     48 45 4C 4C CF
```

The big difference is the last character has its 8th bit set high. It wasn't being used for anything else. Now the end of the string can be recognized. This method is used to store file, directory and module names. There are other ways to keep track of the length of strings in assembly language. I would suggest getting a good book on assembly language and studying it. Also, study other people's assembly code for ideas.

Different schemes in different languages have been devised to store and access strings. Basic09 lets you to set aside an area for string data. The way to do it is to dimension the string with a statement like:

```
     DIM name:STRING[40]
```

This tells Basic09 to reserve 40 bytes in the data area for a string called "name". Now the only thing left is tell how long is the string. If everytime we wanted to print the string, we had to output 40 characters and the item we were storing in it was something less, we would end up printing our string plus whatever garbage it took to make up 40 characters. To take care of this Basic09 puts a marker at the end of the data to signal the end of data. If in a program we had:

```
     name:="COLOR"
```

the data would stored as

```
     43 4F 4C 4F 52 FF
```

When using the data in "name", BASIC09 will recognize the the $FF and use only the first 5 characters. Should our data fill the entire area allocated to it, the marker will not be used.

The manipulation, storage and usage of string data is an important aspect of programming.

C language uses a similar method. To create a data area called "word", our program would contain:

```
     char word[25];
```

This provides space for 25 characters. C uses a marker to denote the end of a string. The line

```
     strcpy("DISK", word);
```

copies the word, "DISK", to our data area called "word". Its storage would look like:

```
     44 49 53 4B 00
```

The $00 is the ASCII null character and used as a terminator for strings in C Language. Unlike Basic09, most C functions look for the null character at the end of a string. C passes the location to a function by a pointer. The function has no way of knowing how long is the data area. So it looks for the the null terminator. Therefore it is best to consider the last "char" of the data area for the terminator.

Pascal stores string data in a character array. To provide space for character data, we can enter a line in the variable declaration section of the program.

```
     animal: array [1..10] of char;
```

This says that "animal" is an array that will hold 10 characters. Either the individual characters must be assigned to each character in "animal" or it must be set equal to an equivalent size area. To assign the word "CAT" to line, we enter:

```
     animal[1]:='C';
     animal[2]:='A';
     animal[3]:='T';
```

or

```
     animal:="CAT       ";
```

The first method puts "CAT" in the first three characters

Different schemes in different languages have been devised to store and access strings.

and leaves the rest of the string untouched. So it is anyone's guess what is in the last 7 characters. The second method puts in "CAT" and 7 spaces to fill the rest of the string. If we were to print "animal" with a statement like:

```
     writeln(animal);
```

all 10 characters would be printed, no matter what they contain. A solution is to create a pascal record. It might appear:

```
     type
        string = record
           size: integer;
           letter: array [1..80] of char
        end;
     var
        animal: string;
```

Now we have a complex variable who can store data by character and size. I our previous example, the character of animal.letter would still be "CAT", but size would be 3. To write this one out we could use:

```
     for i:=1 to animal.size
        write(animal.letter[i]);
     writeln;
```

This time only three characters are printed and the final writeln forces a carriage return.

The higher level languages offer several different methods for the management of string type data. Basic09 is perhaps the easiest to use. Once you allocate space for a string, you need only use it. In C language you have a greater challenge. You must be certain that the string is given enough space and a null terminator is present. In Pascal, you create an array to hold string characters and are responsible for keeping track of its size. At the lower levels, in assembly language you are pretty much in charge of things. OS-9 use the the 8th bit of that last character to mark a string's end. But you can for your programs devise your own method. Perhaps you'll put a marker at the end of each string or maybe keep track of its size. The choice is yours.

CHECKING ACCOUNT BALANCING
MADE EASY

This is the January issue and what better time to start keeping better track of your checking account balance. This months program is in Pascal. And yes it does make use of strings! I wrote the program to help me reconcile my checkbook when I receive my monthly statement from the bank. The program has 5 basic parts: the starting balance, deposits, outstanding checks, outstanding debits and the final balance. Deposits are stored as a total of all deposits made. Checks are stored in an array called "checkrecord", which includes its number and amount. Debits are anything that remove money from your account and isn't a check. Some are service charges, automatic withdrawals and transfer of money between accounts. Debits are stored in an array of "debitrecord", which includes date, name and amount. The final balance is the starting balance plus deposits, minus checks and debits.

You might notice the handling of string data. The variable "name" in "debitrecord" is a string of 35 characters. First, it is set equal to 35 spaces. Next I read in a character at a time into it, until an end-of-line is read. If less than 35 characters are read the remaining ones are spaces since the array of "name" was first initialized to spaces.

Back to the program! After all the data is read, a short summary is printed to the screen giving the starting balance, total deposits, total checks, total debits and the final balance. Every variable is followed by a ":8:2". This tells Pascal to use a field that is 8 characters wide with 2 places following the decimal. This will cause your dollar amounts to printed in nice, tabular form. You are also give the option to direct a more detailed printout to a different path. The detailed printout includes all the information previously obtained. The output can be to any path you want. Perhaps it is /P for the printer, /TERM for the terminal or some file name for the disk.

The line

```
     rewrite(hardcopy, pathlist);
```

opens the pathlist. I didn't use a "close" statement, since it will be automatically closed at the end of the block. It would be easy to draw an analogy to OS-9's concepts of paths. "Hardcopy" might be thought of as the path, "pathlist" is its name, and "rewrite" as the open. But there is a much deeper meaning. "Hardcopy" is a special variable called text. Text is nothing more then an "array of char". In theory it can be endless in size. In reality it is limited by the constraints of the media. Also, "hardcopy" is declared in the program heading, along with input and output. The rewrite command places the files pointer at its beginning. The writeln puts characters into the text and moves the file pointer forward. The statement

```
     writeln(hardcopy, .....);
```

puts characters into the file "hardcopy". We could use a similar statement for the standard output and input, as well as, the error output. They would appear:

```
     readln(input, .....);
     writeln(output, .....);
     writeln(syserr, .....);
```

The three standard paths are understood by OS-9 Pascal. In fact a simple readln and writeln could be substituted for the first two examples. This also applies to uses for read and write.

CORRECTION!

On more thing about last months column should be mentioned. There was a small error in the program, dnc. It would not effect its operation, but you may want to make the change. In the function, "help()" the first line read:

```
     printf("Dname <device name>\n");
```

The problem is the programs name is dnc. It was originally Dname, but later changed. I changed this line to:

```
     printf("Usage:\n");
     printf("   dnc <device name>\n");
```

Physically the program will work, but you might want to make this change.

That is all for this month. Study the use of strings and try the Pascal program, checking. There is so much to Pascal that it would be impossible to discuss it all in one column. In fact it would take an entire course. If you are interested, many colleges and junior colleges offer courses on Pascal. Also, get all the literature you can about it. There are many good books around. In future I will try an discuss more about it. Until next time, have fun!

- - -

```

program checking(input, output, hardcopy);
{ This program is intended to help balance a
  checking account, especially after receiving
  the monthly statement. }
type
   checkrecord=record
      number: integer;
      amount: real
   end;
   debitrecord=record
      month:integer;
      day:integer;
      name: array [1..35] of char;
      amount: real
   end;
var
   check: array[1..50] of checkrecord;
   debit: array[1..20] of debitrecord;
   pathlist: array[1..40] of char;
   hardcopy: text;
   balance, oldbalance: real;
   deposits, entry: real;
   answer: char;
   j, count: integer;
   totalchecks: real;
   totaldebits: real;
   getchecks: boolean;
   getdebits: boolean;
   checksize: integer;
   debitsize: integer;

begin { main program }
   writeln('Checking Account Balancer');
   writeln;
   write('Enter starting balance: '); prompt;
   readln(oldbalance);

   { get any deposits made }
   deposits:=0;
   entry:=-1;
   writeln('Enter deposits, a 0.0 when finished');
   while entry<>0 do
   begin
      write('Deposit: ');prompt;
      readln(entry);
      deposits:=deposits+entry
   end; { of while }

   { find the total checks written }
   totalchecks:=0;
   count:=1;
   writeln;
   writeln('Outstanding Checks');
   writeln('Enter a 0 for number when finished');
   writeln;
   getchecks:=true;
   while getchecks=true do
   begin
      with check[count] do
      begin
         write('Number: ');
         prompt;
         readln(number);
         if (number <>0) and (count<=50)) then
         begin
            write('Amount: '); prompt;
            readln(amount);
            totalchecks:=totalchecks+amount;
            count:=count+1;
            writeln
         end { of if }
         else
         begin
            writeln('Check entry terminated');
            getchecks:=false;
            checksize:=count-1
         end { of else }
      end { of with }
   end; { of while }

   { Find any other debits }
   writeln;
   writeln('Debits to Checking Account');
   writeln('Enter 0 0 for month and day to terminate');
   writeln;
   getdebits:=true;
   totaldebits:=0;
   count:=1;
   while getdebits do
   begin
      with debit[count] do
      begin
         write('Month, Day: '); prompt;
         readln(month, day);
         if ((month<>0) and (day<>0) and (count<=20)) then
         begin
            name:=' '
            j:=1;
            write('Name: '); prompt;
            while not eoln do
            begin
               read(name[j]);
               j:=j+1
            end; { of while }
            readln; { move past eoln }
            write('Amount: '); prompt;
            readln(amount);
            totaldebits:=amount+totaldebits;
            writeln;
            count:=count+1;
         end  { of with }
         else
         begin
            writeln('Debit entry terminated');
            getdebits:=false;
            debitsize:=count-1;
         end { of else }
      end { of with }
   end; { of while }

   { compute current balance }
   balance:=oldbalance+deposits-totalchecks-totaldebits;

   { write out short summary to screen }
   writeln;
   writeln;
   writeln('Starting balance:   ',oldbalance:8:2);
   writeln('Total deposits:     ',deposits:8:2);
   writeln('Total checks:       ',totalchecks:8:2);
   writeln('Total debits:       ',totaldebits:8:2);
   writeln('Final balance:      ',balance:8:2);

   { Find out if a detailed printout is wanted }
   writeln;
   write('Do you wish a printout? Y/N   '); prompt;
   readln(answer);
   if ((answer='Y') or (answer='y')) then
   begin
      pathlist:=' '
      write('Pathlist: ');prompt;
      j:=1;
      while not eoln do
      begin
         read(pathlist[j]);
         j:=j+1
      end; { of while }
      readln; { move past eoln }
      rewrite(hardcopy, pathlist); { open pathlist }
      writeln(hardcopy);
      writeln(hardcopy,'      CHECKING REPORT');
      writeln(hardcopy);
      writeln(hardcopy,'Starting balance:  ',oldbalance:8:2);
      writeln(hardcopy);
      writeln(hardcopy,'Total deposits:    ',deposits:8:2);
      writeln(hardcopy);
      writeln(hardcopy,'CHECKS');
      writeln(hardcopy,'Number   Amount');
      for j:=1 to checksize do
         with check[j] do
            writeln(hardcopy, number:6,'   ',amount:8:2);
      writeln(hardcopy);
      writeln(hardcopy,'Total checks:      ',totalchecks:8:2);
      writeln(hardcopy);
      writeln(hardcopy,' DEBITS');
      writeln(hardcopy,'Date   Name                                        Amount');
      for j:=1 to debitsize do
         with debit[j] do
            writeln(hardcopy,month:2,'/',day:2,'  ',name:35,'   ',amount:8:2);
      writeln(hardcopy);
      writeln(hardcopy,'Total debits:      ',totaldebits:8:2);
      writeln(hardcopy);
      writeln(hardcopy,'Final balance:     ',balance:8:2);
      writeln(hardcopy)
   end { of if }
end. { of main }
```

ADA^{R} And The 68000

by
Theodore F. Elbert
The University of West Florida
Pensacola, Florida 32514

PART 9 - ADVANCED TASKING FEATURES

In Part 8 of this series the basic tasking structure of the Ada language was discussed. The concept of task synchronization by means of the rendezvous mechanism was introduced and illustrated by an example. For the concept of synchronization by rendezvous to have the flexibility required for effective concurrent programming, capability beyond that of the simple rendezvous is required. Often, it is necessary to select from a number of accept statements, depending upon which entry call occurs first. In other instances, some kind of timeout mechanism is necessary, so that a task does not wait an excessive amount of time either at an entry call statement or at an accept statement. Nor is the mutual exclusion problem completely solved by the simple rendezvous, since concurrent calls can be made to the same task if more than one entry is provided into the task.

The extension of task synchronization features beyond those provided by the simple rendezvous is obtained through the use of the select statement, which can be used in the calling task in conjunction with the entry call statement, in the called task in conjunction with one or more accept statements, or in both the calling and the called task. In the calling task, the select statement permits the use of the timed entry call, by which a calling task can timeout and continue execution if a rendezvous cannot be effected within a specified time interval. In the called task, the select statement provides for the use of a selective wait, by means of which a called task can choose from among any number of accept statements in order to find a waiting entry call.

To illustrate the selective wait use of the select statement, the following example is provided.

```
task GET_DATA is
    entry INERTIAL(DATA : out STATE);
    entry DOPPLER(DATA  : out STATE);
    entry AIR_DATA(DATA : out STATE);
end GET_DATA;

-- This is a task specification that declares
   three entries. The type STATE is presumed to
   be previously declared and visible to the
   task.

task body GET_DATA is
          .
          .
          .
begin
          .
          .
          .
select
    accept INERTIAL(DATA : out STATE) do
       -- sequence of statements of accept
            statement
    end INERTIAL;
       -- sequence of statements of accept
            alternative
or
    accept DOPPLER(DATA : out STATE) do
       -- sequence of statements of accept
            statement
    end DOPPLER;
       -- sequence of statements of accept
            alternative
or
    accept AIR_DATA(DATA : out STATE) do
       -- sequence of statements of accept
            statement
    end AIR_DATA;
       -- sequence of statements of accept
            alternative
end select;
          .
          .
          .
end GET_DATA;

-- Shown in this body is a single accept
   statement with three accept alternatives.
```

When the execution of the task reaches the reserved word select, a determination is made of which, if any, of the three accept statements have entry calls waiting. If there is an entry call waiting, the rendezvous is effected immediately by that accept statement and the sequence of statements of that accept statement is executed. Upon completion of this sequence of statements, the rendezvous is completed and the parameter DATA is passed to the calling task, which then resumes execution. At the same time, the sequence of statements of the accept alternative begins execution. When the execution of this sequence of statements reaches the reserved word or, control is transferred to the end of the select statement, indicated by the **reserved words end select.**

If more than one accept statement have entry calls waiting, then an immediate rendezvous is effected with one of them, but the language rules do not specify how this selection is made. On the other hand, if none of the accept statements have entry calls waiting, then the execution of the task is suspended awaiting an entry call to one of the accept statements. When such an entry call is made, the task resumes execution and an immediate rendezvous is effected. The reader should note, however, that when execution of a task is suspended, the computational resources assigned to the task may be returned to the run-time support system for further allocation. Thus, resumption of execution of a suspended task may imply raising the task from the suspended state to the ready state, where it must then await allocation of computational resources. Such effects are implementation dependent.

The selective wait is even more flexible than is implied by the example above. In general, each select alternative consists of an optional **when** condition, followed by either an accept alternative, a delay alternative, or a terminate alternative. The **when** condition evaluates a boolean expresion and, if the result is TRUE, the select alternaive is said to be open and functions as though the **when** condition were not present. If the evaluation is FALSE, the select alternative is said to be closed. No rendezvous is possible through a closed select alternative. The select statement may also have an else part, which is executed if an imediate rendezvous is not possible. The delay alternative, if present, is executed only after a specified time delay, and only if a rendezvous through an open accept alternative is not effected prior to expiration of the delay. As an example of the use of **when** conditions, consider the following task declaration that implements a simple binary semaphore.

```
        task BIN_SEMAPHORE is
            entry WAIT;
            entry SIGNAL;
        end BIN_SEMAPHORE;

        task body BIN_SEMAPHORE is
            BUSY : BOOLEAN := FALSE;
        begin
            loop
                select
                    when not BUSY =>
                        accept WAIT do
                            BUSY := TRUE;
                        end WAIT;
                or
                    when BUSY =>
                        accept SIGNAL do
                            BUSY := FALSE;
                        end SIGNAL;
                end select;
            end loop;
        end BIN_SEMAPHORE;
```

The purpose of the semaphore is to protect some critical resource from concurrent use by more than one task. For example, all Ada subprograms are reentrant--that is, more than one concurrently executing task may be executing a subprogram simultaneously. A semaphore may be used to protect a procedure from this kind of simultaneous execution if mutual exclusion is required. If the procedure name is CRITICAL, then all calls to procedure CRITICAL would be implemented as:

```
        BIN_SEMAPHORE.WAIT;
        CRITICAL;
        BIN_SEMAPHORE.SIGNAL;
```

In this manner, a call to CRITICAL can be effected only if the entry call WAIT results in a rendezvous which, in turn, sets the value of BUSY within the semaphore task to FALSE. With BUSY set to FALSE, the WAIT accept alternative is closed, and thus any other call to WAIT will cause both the calling task and the semaphore task to be suspended. Upon return from procedure CRITICAL, an entry call to SIGNAL results in an immediate rendezvous, since SIGNAL is an open accept alternative. This rendezvous assigns the value FALSE to BUSY, thus opening the WAIT select alternative. Therefore, if one task has completed a rendezvous with entry WAIT, other calling tasks queue in a first in-first out order at the WAIT entry until the first task completes the SIGNAL entry.

A calling task effects an entry call through the execution of an entry call statement. When an entry call statement is executed, an immediate rendezvous is effected if the called task is waiting at a corresponding accept statement. An immediate rendezvous may be impossible for one of the following reasons:

- The called task is not waiting at a corresponding accept statement.

- The called task is waiting at a select statement containing a corresponding accept statement, but the accept alternative is closed.

In either case, the entry call is queued awaiting the availability of the called task at a corresponding accept statement, or at an open accept alternative containing a corresponding accept statement. Just as a timeout protection feature is needed for the selective wait, so also is this feature required to protect the entry call statement from excessive delays, or possibly from a deadlock condition. This timeout feature is also provided by the select statement through the use of the timed entry call. For example, the timed entry call

```
        select
            PRIMARY_LOOP.RADAR_INPUT(RADAR_DATA);
            -- optional sequence of statements
        or
            delay 0.01;
        end select;
```

will suspend the calling task for up to ten milliseconds awaiting an available rendezvous, after which time the entry call will be cancelled and control will be passed to the statement following end select. Alternatively, the select statement may have an else part:

```
        select
            PRIMARY_LOOP.RADAR_INPUT(RADAR_DATA);
            -- optional sequence of statements
        else
            -- sequence of statements of the else part
        end select;
```

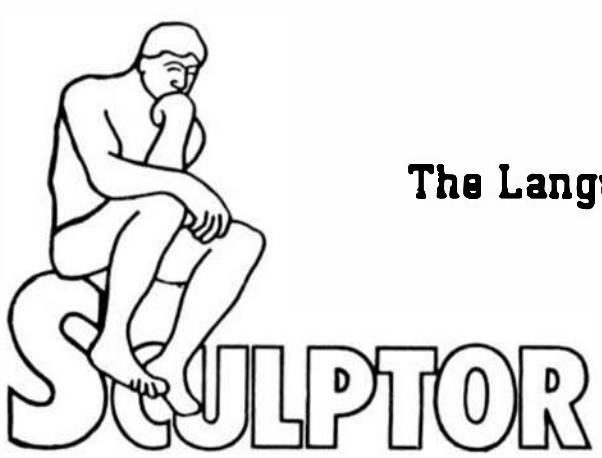
In this case, the sequence of statements of the else part will be executed only if an immediate rendezvous is not possible. Typically, the sequence of statements of the else part would contain some routine background function--perhaps some self-diagnostic program. It is also possible to include a sequence of statements in the delay alternative shown above, in which case the sequence of statements will be executed only after a ten millisecond delay, and only if a rendezvous is not effected during that ten milliseconds.

The concepts outlined here imply a very powerful and versatile capability within the Ada language for implementing programs with concurrently executing tasks. It should be noted, however, that the scheduling of tasks within a program is accomplished by the run-time support system, which may--and probably will--vary from implementation to implementation. For example, different run-time systems may support different numbers of processors, or a single processor system may provide different degrees of multiprogramming. There are other areas of concern that have not been addressed here. Some of these areas of concern relate to task priorities, shared data, and deadlock avoidance. The net result is that the concurrent processing features of an Ada program may not be portable. The programmer can use some discretion to minimize these concerns by using the rich concurrent processing features of the language to explicitly avoid potential problems. For example, the use of shared data should be minimized. Deadlock prevention should be specifically designed into the program, rather than having it depend upon implicit language features that may not be portable. Program correctness should not depend on the relative speed of execution of concurrent tasks. Circular dependency among tasks should be avoided. Finally, some care should be taken in selecting the run-time support system. They are not all the same.

This discussion of tasks concludes the presentation of Ada language features. The concluding part of this series will review the current status of Ada compilers and Ada support software, with particular emphasis on the role of the 68000 microprocessor in the developing Ada scene.

R Ada is a trademark of the U.S. Department of Defense.

NEXT: Ada Compilers and Software.



The Language for Non-Programmers

A Review

By: David Lewis

Formerly SAGE
The LANGUAGE for Non-Programmers

Are you spending lots of time and money trying to get the software that's "RIGHT FOR YOU"? Finally, a tool has been developed that will soon become as indispensable to anyone who owns a computer as a screwdriver is to anyone who works on one. At last 68XX or XXX users there is SCULPTOR a fabulous new software development system that allows even beginners to produce sophisticated data management software quickly and easily. Soon to become the boom to the 68XX market that dBASE II was to the IBM PC market and this new versatile system runs on their machine as well as it runs on ours (of course there are limitations PC users must endure which thank goodness don't apply to us).

The most powerful and useful aspects of SCULPTOR is it's file handling characteristics. By using the Index Sequential Access Method of record storage and retrieval, coupled with modern B-Tree key files, you can experience extremely fast maintenance free data handling. The key to getting the most out of ISAM files is just that, 'the key'. Contemporary key files are static in that they don't always reflect a true relationship with the master data file. In creating them the master file is read, keys are sorted and written sequentially into a key file along with the master record number. To search, the middle of the key file is read and tested for less than, equal to, or greater than. Then as a result if the record isn't found, the file is divided into either an upper or lower half and this process repeats.

Many of you have experienced that after many changes in a file, the key file has to be completely rebuilt, which on a large file can mean hours of down time. The B-Tree (Binary Tree) key file can eliminate this because it is dynamic, constantly reflecting the current condition of the master file. Very simply each key in a B-Tree file contains the master file key data, and record number as well as the location of the next higher and lower key. As the master file gets larger the key file grows like a tree by sprouting new branches. Likewise, as records are removed or changed the existing limbs attached to them are grafted onto the limb before them. For a more detailed explanation refer to Donald Knuth's "Art of Computer Programming".

As you can see, this powerful access method can substantially reduce search time, eliminate the need to rebuild key files except in case of disk corruption, allow files to be automatically sorted in several ways at the same time, and all of this using simple easy instructions. The use of this type of file handling structure is important in SCULPTOR'S ability to automatically produce true multi-user software. You may have bought the 68XX machine because that is what you needed but unfortunately didn't get it. The available software has not utilized this capability to anywhere near the extent that SCULPTOR has. Creating and building your files are also fast and easy! Decide what data you want to utilize, describe it to SCULPTOR, then tell it to make a screen form program for you automatically. Within a blink of an eye SCULPTOR designs your own custom-made program for entering, searching, modifying, and deleting data. Select one more option in the SCULPTOR main menu and you have a written report program. This sounds like Alice in Wonderland, but truthfully it's literally that simple!

With the power of SCULPTOR an individual with average computer exposure can do some amazing things for themselves. For those that wish to utilize SCULPTOR to it's fullest capabilities there's just not enough room here to cover it all. I do hope to be writing more on the subject in future issues so stay tuned. SCULPTOR is a Fourth Generation Language. Now I'm not sure if that means it has a 4th grade education or if it's great, great, grandfather was an adding machine. Anyway in comparing this language to other languages like BASIC, I've found that actual programming time with SCULPTOR is substantially reduced. Using BASIC, I spend approximately 20% of my time actually developing the code to accomplish the program directive while 80% of the time is spent making up screens or formatting printouts or doing other machine dependent tasks.

These time consuming functions have little to do with the idea of the program but are a must for ease of use of the program. It is in these areas that SCULPTOR can really help. By virtually eliminating these machine dependent tasks and allowing the programmer freedom to concentrate on the actual problem at hand - it is possible to develop extremely fast and efficient software in about one fourth the time. Let me give an example: Consider some of the problems one might encounter in writing a typical business program. 1. Handling and coding for different terminal types. 2. Developing usable screen forms. 3. Designing multi-uer file structure. 4. Coding to achieve proper record locking and prevention of file corruption by improper locking. 5. Verifying, changing, and formatting data. 6. Determining actual disk record numbers quickly. 7. Writing print routines with headers, footers, proper spacing, and totals. 8. Utilizing enhanced printer and terminal functions. 9. Producing neat readable, easily debuggable code. These serious problems are commonly involved along with many others in any quality program. SCULPTOR handles these aspects of your program efficiently and quite automatically by: 1. Terminal types and codes are loaded from a terminal file and are used accordingly to the terminal in use. 2. Screen forms can be done completely automatic or on a row and column selection. 3. The data files are all created for multi-user access. 4. Record locking also can be handled automatically or under program control and file corruption is virtually impossible except for something like a power failure. 5. Extensive data verifying, changing, and formatting is done automatically, even more is available under program control. 6. Disk record number selecting is invisibly to program. 7. Easily definable and/or automatic centering, spacing, headings, totals and more for reports. 8. Enhanced printer as well as terminal functions are available for easy use. 9. SCULPTOR is an

English type of language that is both easy to read and easy to write. I have written an order processing system that involves all of the above mentioned features of SCULPTOR and some others for anyone interested.

SCULPTOR is a dynamite package; however, I encountered a few problems at the start-up stage. First, the reference manual does not have an index. As I was reading there were explicitly detailed examples and instructions but when I started coding, referring back to them was time consuming and frustrating. No index! Now I'm sure the people who did the writing didn't mind thumbing through for their answers, but for that matter they probably didn't need the manual anyway. Second, the section on bringing up the system is vague. Be sure to request start-up information for your specific OS with the reference manual. Third, the system does not allow opening a master file without a key file or additional key files without each having separate and possibly empty master files. This means each file occupies 2 channels allowing a maximum of 6 files to a program (UniFLEX). This problem comes up when using multiple cross reference files for a single master file. The program only uses the cross reference key file so the accompanying main file is empty occupying a channel and disk space for no value.

I mentioned in the beginning about the effect this product might have in comparison with dBASE II. In addition to expanding the market base for 68XX systems there are a number of other comparisons. In file and record structure dBASE allows use of only 2 files, each of which is limited to a maximum of 32 fields and a total of 1000 characters per record, allowing only 65535 records to a file. SCULPTOR on the other hand has no limits as to number of fields or size of records. The limit to records in a file is a modest 17 million or the size of your storage medium which ever is smaller. In fact most limitations to SCULPTOR are machine set such as number of files in a program (UniFLEX and OS9 is 6, Quix is 8, and UNIX 8-16). In dBASE all numbers are represented as floating point and all data is stored in ascii format. SCULPTOR's data is stored as represented; Integers 1,2 or 4 bytes, 8 byte floating point, 4 byte date variable, and 0 to 255 byte alphanumeric variable. This allows great storage utilization. Other points to mention are the dBASE allows only a maximum of 64 different variables in a program, SCULPTOR has no limit. dBASE limits index key size to 99, SCULPTOR's limit is 160. dBASE makes no provision for file protection in multi-user operation, I've already mentioned SCULPTOR's super job here. dBASE does not automatically warn users about inserting a duplicate record, SCULPTOR does. Now an interesting difference that swings both ways, dBASE does not allow statement labels or goto's and gosub's. Many of you, especially those used to programming in BASIC, might ask how this could possibly be anything other than a negative. Well dBASE is a structured language using top down program flow, each section and even loop commands have a definite beginning and ending point with no early exit. This allows complex programs to be broken down into independent steps, and prevents changes made in one part from affecting other parts of the program. Couple this with the many good debug tools available to dBASE and if used properly, programs are easy to follow and easy to correct.

On the other hand SCULPTOR is also designed as a structured language, however line labels and branch statements have been included. SCULPTOR does not have any type of internal debugging tools and if your program is jumping all over the place, then finding and correcting errors can be very difficult plus these corrections can cause other problems elsewhere. Programmers used to BASIC need to keep this in mind and it might do well to study the approach used by dBASE. One last comparison I wish to make involves interfacing with outside files. Two different approaches have been taken between these languages, dBASE allows directly opening, reading and writing to what is called foreign files. SCULPTOR on the other hand allows use of outside files only through re-direction of input. I like dBASE's method better because it allows more program and operator control.

I personally think SCULPTOR has the potential to really boost 68XX(X) systems to where they ought to be. I hope it's designers will continue to improve and enhance it and I hope software companies and system owners will fully utilize it. I am including a quick reference guide that you may wish to keep handy. Please address questions or corrections to this magazine in care of David Lewis. I will attempt to cover them in future additions. Thank you

Program Descriptions

cf	:Compiler for 'sage' programs. (files with .f extension)
cr	:Compiler for 'sagerep' programs (files with .r extension)
decprinter	:Decodes printer parameters into reusable file or to screen
decvdu	:Decodes CRT parameters into reusable file or to screen
describe	:Create or edit file record structures
kfcheck	:Checks file integrity and counts records.
kfdet	:Reports details of key and record length and the depth of index usage.
kfcopy	:Re-creates a keyfile retaining existing data (Useful only after extensive deletions)
kfri	:Rebuilds the index in the case of severe file corruption following a system crash
lcf	:Changes date, error, other parameters in run time programs
newkf	:Creates a new empty keyfile
reformat	:Reformat a keyed file
sage	:Screen form interpreter.
sageform	:Print screen form for documentation
sagerep	:Report generator
setprinter	:Setup printer parameters
setvdu	:Setup CRT parameters (NOTE setprinter and setvdu may be fed file created by decprinter or decvdu respectively)

Setting up the System

1. The Language Configuration Program:

Most of the SCULPTOR prompts and date formats are configurable according to any desired as long as length and some other constraints are followed. Below are the standard text and formats along with number of characters available to install change in.

++lcf (pathname of sculptor program) ie. lcf /bin/sage

Text	length	
Notes		
dd/mm/yy	10	mm/dd/yy or yyyy
.	1	Decimal point
,	1	Comma
[	1	Box Delimiter
]	1	" "
Which option do you require?	38	Option prompt
No such record	18	default oer error
Record already exists	22	" re "
Waiting...	14	" riu "
No record selected	22	" ors "
(y/n)?	8	Prompt prompt

(Note on the prompt anything you use to replace the y or n must line up with the y and n in the original)

return with no gosub!	25	
Illegal arithmetic!	23	
Stack overflow	19	
Described record length > actual	37	
7	1	Chr size (7 or 8)
Bad input data	18	
r	1	<CR>

(note sagerep will append a <CR> at end of reports to accommodate the advanced spooler if this is not desired put a ´x´ here)

II. Setting Up Terminals

A number of programs and files are provided to setup terminal and printer parameters. All of the screen programs running under UniFlex access CRT parameter files in the /gen directory. If your system is using only 1 terminal or 1 type of terminal and if a file exists in the supplied disk for your terminal then copy this file into the /gen directory and name it vdu (/gen/vdu). If you are using more than 1 type of terminal in a multi-user environment SCULPTOR programs will pickup your tty number, strip off leading zero´s, and append it onto vdu to locate the parameter file. If the parameter files for your terminals are supplied then determine which port each one is connected to and copy the par. file into (/gen/vdu´ttyno´). /gen/vdu0 for port 0 /gen/vdu1 port 1 and so on.

If a parameter file is not supplied for your terminal grab the manual for your terminal, copy what you think might be the closest example in the ´vdus´ directory into a temporary file and execute this command "decvdu (temp filename) >(vdu.txt)" you can call ´vdu.txt´ anything you really want. Then ´edit vdu.txt´. The descriptions along with the hex codes will be in the file. Refer to the SCULPTOR manual for more detailed descriptions and change the hex codes to those for your terminal. After you have saved the file, then exec

"setvdu /gen/vdu(ttyno) <vdu.txt"

substitute your port number for ttyno or leave off, if all terminals are the same. By redirecting the input from your text file, all you have to do is change only the problem and do it again, if it does not work right the first time .

III. Setting Up Printers.

The sagerep program reads the printer parameter file as an argument from the command line ie "sagerep (program name) (printer file) (other arguments)" the printer files are not as complicated and again if an appropriate printer file is not supplied use the "decprinter (parfile) >(textfile)" to create a file to edit then use the setprinter program with redirected input from your text file to create a printer file in the /gen directory with a name you can use on the sagerep command line. Ie

"setprinter /gen/epson <epson.txt"
"sagerep customer.r epson >/dev/ppr"

IV. Getting Started

I am assuming here that you have installed the system using the supplied insert file. A note about using the insert file; check to see that you don´t already have a program called menu in your /bin directory. That seems to be the only common name used by SCULPTOR programs. SCULPTOR programs are in /bin and /usr/bin and there will be a /usr/sculptor directory containing system files and a couple of demo files. At this point I recommend you create your own directory such as /usr/demo. There is a command file in /usr/sculptor called newsys which makes use of the ability for Uniflex command files to utilize arguments.

As a quick illustration of argument passing in command files let´s assume there is a compiled BASIC program that reads a line from the standard input and prints the same line followed by a blank line to the standard output "#0". We will call this program "dblspace" (creative eh!) and set execute permissions "perms u+x o+x dblspace" now utilizing the "page" command with line numbers we want to create a command file that will give a double spaced but correctly numbered printout. We edit a file "edit pgedbl" (1.00=page $1 $2 ^dblspace) when we execute pgedbl the variables $1 and $2 will be replaced with arguments (pgedbl test.txt +lp33 >/dev/spr) "test.txt" will replace $1 and +lp33 will replace $2. Run it and you will get correctly numbered lines double spaced and printed with headers at top of each page (p33 numbers only 33 lines before issuing formfeed). Off the subject? Not really, execute the command file "/usr/sculptor/newsys /usr/demo (or what ever directory you create)" and the argument ´/usr/demo´ is used to create system files in that directory. Now you have a demo directory ready, to start type "menu" and you should be up and running.

SCULPTOR Program Definitions

!box <delimiting characters>
defines non standard characters to enclose screen form input/output areas.
!box <>   !box :   !box " " This sets to space

!depth <integer>
Defines screen depth. Used only where terminal supports feature default 24   !depth 20 sends codes to reconfigure terminal to new size

!file <file identifier> <pathname>
Declares keyed file. Pathname may be omitted if identifier is same as pathname.

!file stock   !file cust customers   !file tax /income_tax

!record <file identifier> <pathname>
Declare an alternative record layout for file identified in !file. Must have same key length, up to 8 !record statements per file.

!file xref !record xref names !record xref phone

!scroll <heading line>,depth
Defines area of column data display. Accessed by variable ´scrline´
!scroll 10,5   (5 rows under heading at line 10)

!temp <name>,<heading>,<type&size>(<dimension>)
Temporary variable used in program. No validation permitted, format only in box def
!temp total,Total,m4 !temp ebal,End Bal,s0
!temp cat,Category,s2(10)

Special temp fields

!temp arg,,s0	Command line arguments
!temp date,<heading>,d4	System date
!temp scrline,,i2	Scroll line variable
!temp systime,<heading>,i4	System time in seconds
!temp task,<heading>,s5	The current task number
!temp time,<heading>,m4	Current time: hours.mins.
!temp tstat,<heading>,i1	Child task termination status
!temp ttyno,<heading>,i1	tty port number
!temp day,<heading>,i1	Day for encdate/decdate commands
!temp month,<heading>,i1	Month " " " "
!temp year,<heading>,i2	Year " " " "

!width <integer>
defines screen width on terminals that support, default 80 !width 82   Sends codes to reconfigure screen to 82 columns

cancel {on/off}
If on, operator can abort with CANCEL KEY. If off can´t

chain <text expression>
Terminates sage and replaces it with a new program.

check <file identifier> [nrs-label]
Checks that a record has been read and not written or cleared

clear <box list>
clears the screen and data values. If box list given clears only those boxes note if no box list given clears all variables in program careful
clear name,code,amt   clear name-amt_due   clear

decdate <expression>
Decode a date into day, month, year components

delete <file identifier> [nrs-label]
deletes the currently selected record from file

display <box list>
Displays current values of specified fields into boxes on screen

encdate <date field>
Encodes current values in month, day, year into date field

end
Terminates statement execution returns control to option prompt

error <text expression>
Displays error message in bottom left of screen

exec <text expression>   or   execu <text expression>
Executes expression as child task exec unconfig and reconfig screen, execu executes without changing screen (unseen)

exit <numeric expression>
Terminates sage program returning control to calling task or system

find <file identifier> [key-field list] [nsr-label] [riu-label]
Searches for first record whose key matches supplied key. Does not requiring an exact match.

gosub <label>
Transfers control to subroutine at line labeled

goto <label>
Transfers control to statement at line labeled

if <expression> then <statement> [:<statement>]
Statement(s) executed only if expression true

input <box list> [bs-label] [eoi-label] [ni-label]
Positions cursor to each box and awaits input, if valid assigns var. If not, sound bell and repeat.

insert <file identifier> [key-field list] [re-label]
Inserts a new record into file checking that key does not already exist

let <field value> = <expression>
set the variable to value   total = qty*price

match <file identifier> [nsr-label] [riu-label]
Return next record whose key matches previous 'find'

message <text expression>
Displays a message in bottom left corner

newform
Clears screen redisplays the screen form used on return from 'exec'

next <file identifier> [nsr-label] [riu-label]
Reads the next record in ascending key sequence from last key by any access cmd

nextkey <file identifier> [nsr-label]
Reads key only for next ascending avoids 'riu'(record in use) error. Gives 'nsr' (no such record) at end of file

prompt <text expression> [no-label] [yes-label]
text centered under menu line with (y/n)? no or yes as entered

read <file identifier> [key-field list] [nsr-label] [riu-label]
Reads record whose key matches exactly

readkey <file identifier> [key-field list] [nsr-label]
reads key only that exactly matches one supplied avoids 'riu'

return
Return control from subroutine

rewind <file identifier>
Repositions file to the start so that 'next' command returns first record in file

scroll [expression]
Resets the variable 'scrline' 0=scrline +1, >0=scrline to value, <0=scrline-value

sleep <expression>
Suspend execution for the number of seconds

testkey <file identifier> [key-field list] [nsr-label]
tests for key that matches exactly. avoids 'riu'

unlock <file identifier>
Unlocks currently selected record

write <file identifier> [re-label] [nrs-label]
writes back and unlocks selected record

- - -

Editor's Note: Due to the many combinations that SCULPTOR allows for screen forms and data file designation, it is impossible to cover all, or even a majority of SCULPTOR, in one setting. However, if enough interest is indicated (letters, etc.) then we will continue this as a mini-tutorial series. So let us know.

- - -

IN THE MILL: Coming soon will be several options for the Mustang-020 68020 SBC. One will be an 8 port expansion kit. This will be mounted similar to the standard 4 port serial (DB25) connector mounting board, now included with each Mustang-020. This will allow a total of 12 serial ports.

The straight serial expansion above is for OS-9 users. The intelligent I/O below will be for our UniFLEX VM version (in development). This version will have a MMU, Memory Management Unit standard.

We will publish in 68 Micro Journal when these will be available.

* dBASE and dBASE II are registered trademarks of Ashton-Tate.

SCULPTOR is available in single and multi-station system configurations - depending on type computer system. See **S.E. MEDIA** catalog, this issue. Please note that dealer prices are also available.

ALSO

******* Only for 68 Micro Journal Readers! *******

SAVE $150.00 Off Any SCULPTOR Development Package

SAVE $50.00 Off Any Sculptor Run-Time Package

Offer Expires January 15, 1986

- - -

EMERALD COMPUTERS INC.

ESB-1 68008 Single Board Computer

by
Frank Henriquez
500 Landfair Ave.
LA., CA 90024

I've wanted a single board 68000 computer for some time, but most of the boards available were too big, or lacked memory and I/O devices. I wanted a 68000 CPU, at least 128K of memory, a floppy disk controller, and at least one serial port. I also wanted an expansion port, so I could add A/D converters, graphics boards, etc. Oh yeah, and I wanted all of this on a board not much bigger then a 5 1/4" floppy disk drive. Resigned to having to build it myself, I was paging through some Magazine ads when I came across the Emerald Computers ESB-1 Single **Board 68008 Computer**. For about $600, the board comes with a 68008 (an 8 bit Data-Bus version of the 68000) running at 8 MHZ without wait states, 128K of on board RAM, two serial ports with baud rates from 50 to 38.4K baud, a parallel printer port, a floppy disk controller (it can connect to most 5.25" and 3.5" disk drives and can handle double sided, 80 track drives) and an expansion bus. The board provides memory decoding and control for another 128K of RAM, and the memory could be expanded to a total of about 900K of RAM. The I/O section is decoded to occupy a 64K block of memory at the top of the address space, just below the 64K block of memory that is set aside for EPROMs. The board also has a powerful monitor program, EMON-68, in EPROM. This monitor includes a small single line assembler, a disassembler, the usual monitor commands, and routines to access the disk drives.

At first, I was less than enthusiastic about the 68008. Since It has an 8 bit data bus, I assumed that it would be half as efficient as a regular 68000 (internally, both CPU's are identical), but I later learned that the 68008 is about 60% as efficient as a 68000. So, since the ESB-1 had everything else I wanted, I ordered the board.

I completed my system with an Apple-type switching power supply, two Teac 55F double sided, 80 track 5.25" disk drives, and a small case to pack everything into. Eventually, I got the ESB-1 up and running, happily displaying the monitor prompt on my terminal.

There are several ways to use the ESB-1. You can use it with the built in monitor exclusively, or you can replace the monitor EPROMS with a FORTH interpreter in EPROM. CP/M 68K is also available for the ESB-1.

CP/M 68K

CP/M 68K comes with the usual CP/M commands (including that horrible line editor, ED), an assembler, DDT, and a "complete" C compiler. So, after I got tired of playing with EMON, I sent Emerald Computers a check for $239 and a couple of weeks later I received CP/M-68K. Depending on the type of disk drive you're using, CP/M 68K can take up to 5 floppies. I received two floppies, one with CP/M 68K and the CP/M utilities, and another floppy with the C compiler. I also received two manuals, one for CP/M and one for the C compiler. For $111 you can get all the systems files you would need to rewrite the BIOS and the boot programs. Emerald has also included a listing of their BIOS on the CP/M system disk.

Booting CP/M is the most annoying feature of the ESB-1. The EMON monitor doesn't have a disk boot command, so you first have to enter about 8 bytes of data into memory. Then you give a "read tracks" command to read in the CP/M boot, and finally, you type in a jump to the boot program which, after a good deal of disk activity, loads in CP/M.

There are many, many feature of CP/M that I dislike, so I'll just describe some of the worst. Apparently, many of the CP/M 68K utilities are written in C, because many of them are enormous. Luckily, the ESB-1 has a very fast set of disk routines, so the overall speed turns out to be about the same as that of Flex.

Disk Formats

The CP/M-68K disk format uses eight 512K byte sectors per track, so a double-sided double-density 40 track floppy will only hold about 320K of data. The limited data capacity of each drive, coupled with the enormous size of some of the CP/M utilities, means that you'll run out of disk space very quickly. The easiest solution is to get 80 track double sided drives. This will give you about 640K per floppy, enough room to fit all the CP/M utilities and the C compiler on one disk.

So, lets say you can live with typing in a few bytes of data each time you boot CP/M, and you have 80 track, double sided disk drives so you have room to work in. You're all set to write software, right?

NO.

Problems with CP/M 68K
...or...
do I really need this?

I don't know how many of you have experience with CP/M, but something you learn to accept with CP/M is the truly useless documentation, although CP/M-68K is an improvement of sorts. The CP/M manual itself is readable, and the documentation for the operating system routines is quite good (it reminds me of the TSC Flex manual). Unfortunately, the C compiler manual is a complete waste of paper.

On the first page it tells you that the C compiler is compatible with the Unix C compiler, and then goes on to say that the CP/M-68K C compiler does not include any floating point routines. However, if you page through the descriptions of the Implemented C routines you'll find most transcendental functions, plus descriptions of several floating point commands! If you read on, the manual does mention that there are two floating point routines available, the IEEE standard floating point library and the faster Motorola floating point library. It tells you that you must specify which library you wish to use, but it does not describe the libraries at all. By now you may be a bit confused, angry, and frustrated (I sure was). I tried reading and rereading the manual, but after a while I just gave up trying to figure out the C compiler. A call to Digital Research (the people that wrote CP/M) was not very helpful; after a long wait on the phone, they told me that they did not speak to "end users"!

I finally did manage to compile a couple of small test programs, but the more I worked with it, the less "Unix Compatible" it became. I was also treated to a seemingly endless series of disk accesses as the compiler compiled, assembled, and linked the programs. If you're really lucky, you won't get an error message two or three minutes into the compilation; but, that's one of the problems with ANY Compiler.

Well, we've established that the C compiler is basically useless, so what about some of the other programs? Another winner is ED, the resident line editor. There's nothing wrong with this program, it works. What is wrong is that it is a line editor! I was expecting something a bit more modern. A rudimentary screen editor would not have been too difficult to write.

What about the assembler? Well, at least the assembler works well, and it's probably the saving grace of the whole operating system. It's fast (by CP/M 68K standards) and it's easy to use, with lots of options.

As you can guess, I'm less than happy with CP/M 68K. The operating system is functional, the assembler is good, and if you can get or write a simple screen oriented editor, CP/M-68K might even be useful. If Digital Research would rewrite most of the utilities in assembly language and make them smaller, and if they fixed the C compiler documentation enough to make it usable, then I'd say that CP/M-68K would be worth the asking price (but then, THEY have stated that they are not supporting CP/M-68K any more; they are now supporting the "Concurrent" Operating System, instead). As it stands, it's barely worth a tenth of it's price. However, if you're only interested in assembly language programming, you might find CP/M-68K useful.

What about the ESB-1? Don't let my negative view of CP/M-68K scare you away from this little gem of a computer (there's a pun there, somewhere). It's a well designed board, and the people at Emerald decoded the I/O and EPROM sections so as to maximize the amount of RAM that could be added to the system. Since it has all the address, data, and control lines available on two connectors, it would be quite simple to expand the computer (you could probably interface it to the SS-50 bus, if you're so inclined).

OS-9/68K should become available for the ESB-1 soon, although it will require that you expand the memory to at least 256K. I've heard that some of OS-9's utilities are written in C. I have nothing against high level languages; I just think that operating systems, languages and utilities should be written in assembly language to maximize speed and minimize size (unless you have CPU Speed and Memory to BURN!).

Overall, I'd give the ESB-1 a four star rating. CP/M 68K only gets two stars, thanks to the excellent assembler.

EMERALD Computers address is:
4168 S.E. International Way, Suite D203
Milwaukie, Or 97222
See their ad on page 52 this issue.

Add A Hard Disk Drive To Your FLEX or OS-9 Single Board Computer

By Jon H. Larimore
With Michael Birdseye

"20,096 sectors! Wow!!" That's what you'd have heard had you been here in my computer room when I saw that number on my terminal a couple weeks ago. I had just spent a seemingly endless 45 minutes waiting for the format program to finish running on the new outboard 5-Mbyte hard drive we had connected to my little 6809 FLEX single-board computer. Although I had been considering the possibility for some time, (forever?), we had only actually done something about this project a couple of weeks ago when I acquired the disk drive. We'd been working on it for a couple of evenings lashing it together, and it was still lying kludged all over my computer table in naked pieces talking to each other through cable that had to be too long to let it work. Mike had gone home depressed the evening before because it didn't work, and I had said "But Mike, I KNEW it wouldn't. Look at this mess!" But tonight, work it did! And perfectly!

You just might be able to do it too. And perhaps for less than half of the $2000.00 or so you'd have to spend for a complete commercially assembled system.

THE SCARY PART..

Now, before you jump into this headlong, there are a couple of caveats I should mention here, so listen up:

1. The most reliable way to obtain a 6809 computer with hard drive is to buy it commercially assembled. Advertisers in this publication offer excellent versions. They can be easily repaired under warranty by someone else if problems develop. For business purposes or for storage of critical data, that's the best route to take.

2. This article is NOT a step-by-step description of exactly how to duplicate what we did, just a general outline, with necessary specifics.

3. Although this addition is relatively quick, inexpensive, and easy for anyone familiar with the hard and soft realities of the 6809 CPU, it isn't for anyone who doesn't have good eyesight and a steady hand, or who doesn't know which end of a soldering iron to hold, or an RTS from an IRQ. If you haven't worked in assembler or with computer hardware, seek qualified help.

4. Our installation was made on a Peripheral Technology PT-69 computer. Although these procedures should theoretically work with any 6809-based machine, life and computers hold no guarantees. Also, connections may be soldered directly to the 6809 CPU and address decoder sockets. Carefully, carefully.

5. Our source for the disk drive we used (listed at the end of this article) is a surplus dealer whose stock changes quickly. You may or may not be able to obtain exactly the same drive model from them that we used. Most drives will work as mine did, but then, "caveat emptor". Getting it in the package deal from the listed interface and software vendor may be less risky.

Now then, having gotten that out of the way, lets proceed...

WHAT IT IS..

First of all, the "secret ingredients" for this whole project are a neat little interface board and accompanying software first advertised in the September 1985 68' Micro Journal. You may have noticed their ad. A company calling itself "Wellwritten Enterprises" offers an excellent generic package of component parts intended to add a hard drive to an SS-30 system. You may purchase the entire package from them if you wish, including the disk drive, interface card, controller card, software, and cables. Or, optionally, you may choose to purchase only the component parts you need, as I did. They're nice folks to deal with, their prices are right, and their products certainly worked for me. (Yes, I know this isn't an SS-30 machine we're working on here, but more about that later.)

Although I purchased their FLEX driver software because I happen to be nutty about FLEX and STAR-DOS, they'll be happy to sell you equivalent driver routines running under OS-9 Level I or II, or OS-9 68K.

WHAT IT DOES..

As designed by Joe Dubuc for Wellwritten Enterprises, the FLEX system described here uses software which ensconces itself cozily at the top of your user memory, adjusting MEMEND, then, overlaying your existing DOS floppy disk driver jump vectors, re-directs DOS disk calls to itself. It, in turn, using drive tables you set up, determines whether DOS is attempting to call your old floppy drives or your new hard disk drive, and sends or retrieves data accordingly. Neat.

The software talks to an interface board located at the memory location of your choice. Although this is an SS-30 board, we proceeded, undaunted by the pin connectors at one side, to simply solder individual ribbon cable conductors to the pin locations we needed. Worked fine. A little sloppy, but fine.

The interface board has an industry standard connector which connects via ribbon cable to a Xebec #S1410 Hard Disk Controller board. And it connects via two pairs of standard connectors and two ribbon cables to your hard drive. And, of course, the drive and boards must be powered by a sufficiently husky power supply. (We're talking heavy duty amperage here folks).

WHAT WE DID..

(First, the hard stuff..)

First, we removed the 5-volt regulator from the interface card and connected the SS-30 8-volt line directly to it's 5-volt card bus. (Sorry, Wellwritten, hate to mess up a nice board design.) We did this because we planned to steal power from the PT-69's 5-volt supply to run it, and it was thus unnecessary. At that point, we weren't sure whether we'd wind up putting this card in the PT-69 case or the drive case. (It wound up, as you'll see, in the drive case.)

Next, we fabricated cables sufficiently long enough to lay the cards working side-by-side beside the computer, the hard drive, and a test power supply.

Fortunately, most of the connectors are rather common, with the singular exception of the one used for board and drive power connectors. That one is a sonofagun to find. For the cable from the computer to the interface card, we used 25-conductor ribbon cable, with DB-25 connectors in the middle for convenience, and a few unused leftover conductors. (The computer end of this cable is soldered to the CPU board.) And let's be correct here, the cable with the juice on it, the computer end, gets the female connector. We'd strongly recommend the use of color-coded cable here rather than the grey stuff, makes life much easier. As I said, we stole a ground and +5 volts from the PT-69 power supply. As it turned out, we could have just as easily gotten the 5 volts from the disk drive power supply, but even though the card ultimately wound up in the disk drive case, this scheme resulted in one less connector. (Let's save those pennies, ladies and gentlemen.) We used ribbon cable compression insulation displacement connectors. They sure save a lot of time, soldering, and new words in the vocabulary. Mike says he'd buy a drink for the guy who invented them, but I suspect the way things are today, that would take a delivery truck full of cans of whatever.

SUCCESS IS MAKING THE RIGHT CONNECTIONS...

We connected the ribbon cable from the interface card to the CPU board in the following manner. Although we used at least a couple of feet of cable, and "good design practice" would dictate the installation of line buffers/drivers on the CPU card, and we didn't use 'em, we still didn't encounter any noise or data loss problems. We attached the male DB25 connector to the drive cabinet.

Although we simply soldered the CPU board connections to the backs of the IC socket pins, if your computer uses socketed IC's, you might wish to consider using an alternate method, especially if you dislike soldering in tight places, or if you might wish to disconnect this whole thing later. You might look for IC sockets in sizes appropriate to your 6809 and address decoder IC's which have extended length board pins small enough to plug into another socket. Then, simply make your connections to these "sandwich sockets", and plug them in between the IC's and the existing board sockets. I believe I've also seen IC test adaptors which clip over soldered-in IC's. These might be a removable alternative for unsocketed boards, but I do wonder how reliable they'd be over the long run. Be sure to remember that IC pin #1 is the one beside the little dot, the pin numbers run down that side and up the other, and that the sides are reversed when you view the IC from underneath.

INTERFACE CARD CONNECTOR PINS	FLAT RIBBON CABLE COLOR	CPU BOARD IC PINS
		(74LS138)
CS (Pin #1)	Brown	Pin #12
		(MC6809E)
RST (Pin #2)	Red	Pin #37
R/W (Pin #10)	Green	Pin #32
E (Pin #11)	Blue	Pin #34
D7 (Pin #12)	Violet	Pin #24
D6 (Pin #13)	Grey	Pin #25
D5 (Pin #14)	White	Pin #26
D4 (Pin #15)	Black	Pin #27
D3 (Pin #16)	Brown	Pin #28
D2 (Pin #17)	Red	Pin #29
D1 (Pin #18)	Orange	Pin #30
D0 (Pin #19)	Yellow	Pin #31
RS1 (Pin #20)	Green	Pin #9
RS0 (Pin #21)	Blue	Pin #8
IRQ (Pin #22)	Violet	Pin #3
+5 (Pins #8,9)	Orange & Yellow	+5 bus
GND (Pins #25,26)	Gray & White	GND bus

The ribbon cable from the interface card to the controller card is 50-conductor. The two cables from the controller card to the disk drive are one 34-conductor, and one 20-conductor. Connecting the boards to each other and to the drive was simply a case of compressing the appropriate connectors onto the ribbon cable. You might want to consider just buying 50-conductor cable and stripping it down to the other two sizes you need, or perhaps ordering cables already fabricated from Wellwritten Enterprises.

(Then, the soft stuff..)

For FLEX, Wellwritten supplies a hard disk driver program called SBEDR1.CMD, an initialization program called SFORMAT.CMD, a drive reconfiguration program called SCNFG.CMD, and they also throw in some dandy floppy file management programs which I think I've seen in 68' Micro Journal. As a command file, the hard disk drivers can be included as a STARTUP command and loaded automatically during system initialization.

Note, importantly, that like most of the better 68XX software houses, they gladly and willingly supply source code for these, as a matter of course. If anyone ever asks you why you stick with the 6809, that's one of the best reasons I can think of.

As I mentioned early on in this epoch, having created a mess of cables and boards on the table, when we loaded the software and ran it.. nothing! Well, that's not exactly correct. Actually, it locked up. I think it was then 2 AM, and that's when Mike packed it in and trundled off home. I called it a night too a few minutes later.

As I was drifting off to dreamland the thought struck me that since SBEDR1 appeared to have been initially written for SWTPC Flex 9 (Yes, it's different from my 6809 FLEX Version 3.01, #%$-&$%!!!), perhaps it might be infringing upon some of the memory areas my system had already claimed for its own. Next evening, when I checked, sure enough, one double-byte address register had bludgeoned the end of my floppy disk drivers. I'm using Peter Stark's floppy disk drivers by the way, love 'em, but my version extends to $DFBC, and the CFOMAT controller format routine address register in the SBEDR1 driver software was located at $DF9D. I moved it to $DFBD. Note, however, that this address storage register is expected to be at it's original location by the format program and also by the reconfiguration program, so if you do relocate it too, don't forget to change it in those programs also.

You'll note that the hard disk port address specified by SBEDR1 is $E00C. For the computer described here, this is fine, because it corresponds to the use of pin #12 in the computer's 74LS138 address decoder for sending the chip select signal to the "CS" line of the interface board. Other addresses are available from this decoder chip in this computer, and of course you may need to change this address for other computers.

Having reassembled SBEDR1.TXT as SBEDR1.CMD, I then called it up, and, lo and behold, it gave me "HARD DISK DRIVER LOADED" on my screen. Whoopee!

Now, the little ticklies at the back of the neck, as I entered "SFORMAT" and hit return. Up came a menu of options, and when I selected the "full whammy" (controller format, media verification, and FLEX formatting in one fell swoop), it started counting numbers at the bottom of the screen. And counted.. And counted.. And counted.. . After about twenty minutes, I began to wonder where this would all end, so I got out my little TI Programmer calculator and hastily figured out how many sectors five megabytes really was.

While it was happily counting to wherever, I went downstairs and had a cupa coffee. Later, running Puglia and Taylor's CAT, I was pleased to see that SFORMAT had ended at a surprisingly high number of sectors. As it turned out, we got more than we bargained for. On this particular drive, you appear to actually get more than 5 Mbytes of FORMATTED data storage. That's commendable. And on the one we happened to get, there were NO bad sectors.

Subsequent copies of files from my floppy drives to the hard drive, editing them with Screditor III (another masterpiece, bless the late John Alford's soul), renames, and deletes all confirmed that, as far as FLEX was concerned, this was just another disk drive (albeit a mighty big one, at that).

Mr. Thomas J. Weaver, president of Wellwritten Enterprises, points out an interesting facet of the SBEDRl formatter. Since FLEX can only handle a maximum of 16 formatted Mbytes, by changing a couple of clearly marked lines of code in SBEDRl, it will then automatically subdivide a larger drive into two logical drives. The only limitation to bear in mind when you do that is that any subsequent format will erase to the end of the physical drive (both logical drives).

SO NOW, LET'S MAKE IT LOOK PRETTY..

We already had the cabinet listed in the parts list, so we used it. You could probably do a slicker looking job using one of the "Big Blue" clone hard drive cases listed in surplus catalogs, but this one worked fine, and in fact, matches the putty colored cases of my PT-69 and floppy disk drives rather well. It has a power supply which appears to be more than adequate for the controller card, hard drive, and a floppy drive we installed therein, but you ought to verify it's capability with larger hard drives. I'm told they suck up a bunch of amps during start-up.

I got a friend at work to cut and bend two aluminum mounting plates for the Xebec controller card and the Wellwritten interface card. They're nothing more than flat plates cut to the size of each card, with a 1/2-inch flange bent at right angles along one edge having a couple of holes punched in it for mounting. We punched holes (carefully, very carefully !!) in the interface card and mounted it to it's plate with fiber standoffs and screws, then mounted the Xebec card (it already had appropriate holes) in a similar manner.

For some strange reason, I then spent an incredible amount of time just staring at the cabinet and the boards, trying to determine the most appropriate layout and mounting arrangement. You'd think someone was actually going to peek inside someday a hundred years from now and say something like "My, my, didn't Jon do a neat installation job here". Weird. Anyway, if you use this cabinet, you'll find as we did, that it's designed to hold two full-width drives, so you can install the hard drive in the right-hand panel cutout, a floppy drive (or two) in the left-hand panel cutout, the Xebec controller on it's mounting plate between the right side of the hard drive and the cabinet side, and the SS-30 interface card on it's plate nestled snugly above the power supply, mounted horizontally. The controller card plate screws down to the case bottom to the right of the drive mounting baseplate, with two screws and nuts. The interface plate flange just neatly slips between the power supply heat sink plate and the cabinet back, with no screws at all. Pressure holds it in place, and if you install the DB25 connector I mentioned earlier, in the connector cutout directly above it, that connector will hold the interface card and plate firmly in place. By the way, for best fit in this cabinet, the controller card should be mounted on the side of it's plate toward which the mounting flange bends, while the interface card is mounted on the side away from the flange. I had an odd 3/4-width floppy drive I had obtained (mail-order, of course), for a give-away price some months ago, and I finally found a place to put it, in this case, as the #0 "boot" drive.

AND FINALLY..

You'll note that the system described here does not boot from the hard drive. It boots, as it always has, from drive #0, still a floppy drive. You then use ASN.CMD in STARTUP.TXT, to re-assign the system drive to drive #1, the hard drive.

Judging from Mr. Dubuc's driver software, SBEDRl was designed to do a boot from the hard drive, providing there is a boot routine in EPROM somewhere. I'm using a FLEX auto-start version of the PT-69 PT-MON monitor from Peripheral Technology, and they do sell the source code for that monitor, so I suppose one could include the necessary boot code in a patched version of it, if he desired elegance that much.

After the drive had been running for a day or two, a nasty little screech seemed to come from the cabinet periodically. You know how worn brake drums sound on your car? Well.. I knew it had to be the fan, so I ignored it for a couple of days. Finally, it got the better of me so I opened up the cabinet, and found the sound to be emanating not from the fan, but alas, from the hard drive. After mentally comparing the minimum repair charge for a hard drive against the surplus price I paid for the thing, I lifted the drive out of the cabinet and fired it up naked again. And guess what? I discovered that, somehow, the manufacturer had omitted an obviously vital part of the drive's design - - the Scotch tape. (No wonder they sold surplus for such a good price!)

There's an H-shaped metal doodad in the middle of the circuit board which protrudes down through a circular hole to contact the conical end of the drive shaft upon which the flywheel is mounted. (How's that for a description?). I think it's a static potential discharge device, or something like that. Anyway, it was singing. Loudly. Often. Off-key. When I lightly touched it, it quit. So I stuck a small piece of Scotch tape on it (lowering it's resonant frequency, I guess), and quickly buttoned the whole thing back up again, before anyone saw it. It's quiet now...

Now then, about that huge list of files this drive holds. You'll quickly discover that it's easy to run out of file names. What we really need is a program or patch which creates a number of FLEX directories on each drive, preferably tree-structured. Anyone willing to tackle that one?

Let me know how you make out with your version of this project. Best regards,

Jon (CompuServe #75766,3135)

NO, I DIDN'T FORGET THE PARTS LIST...

Hardware, software:

* The works: A 27-Mbyte WREN hard disk, Xebec S1410 controller, SS-30 interface card, all cables, and software: Price $2850.00

>> Wellwritten Enterprises
P.O. Box 9802-845
Austin, Texas 78766

* Same as above, but with a 5-Mbyte Shugart 604 drive (the one we used):
Same source.
Price $750.00

* Same as above, less the hard drive:
Same source.
Price $600.00

* SS-30 interface card and software only:

Same source.
Price $200.00

* The hard drive: We used a Shugart SA604, but they (and other surplus houses) sell other similar drives. A recent catalog lists a Shugart 12 Mbyte 1/2-Height drive for $399.00, and a Seagate 25 Mbyte 1/2-Height drive for $499.00.

Priority One Electronics #FTSHU604
Price $99.00 (@ $89.00 for 5 or more)
(Technical Manual- FTSHUSA604M: $15.00)

>> Priority One Electronics

* The Xebec Model S1410 Hard Disk Controller:
Price $295.00

>> J.C. Information Systems

* The cabinet (with power supply):
Integrand "Little Board Enclosure" Model #2800
Price $150.00

NOTE: They make several other very high quality enclosures designed for single-board computers and drives which may be better suited to your needs. Call and ask for their catalog.

>> Integrand Research Corp.
8620 Roosevelt Avenue
Visalia, California 93291
Phone 209-651-1203

Eds Note: Intergrand also carries the hard to find drive power connectors. These people are real experts in their field. See their ad on page 61.

* Prefabricated cables :

>> Wellwritten Enterprises
>> Priority One Electronics (check their catalog)
>> Your local parts dealer (perhaps)

* 50-Conductor ribbon cable (color-coded)
Alpha #3581/50

(NOTE: The cable and connectors listed here have a .050" conductor spacing. Other conductor spacings are available. Be sure your cable spacing matches the connectors you use.)

* 34-Conductor ribbon cable (color-coded):
Alpha #3581/34

* 20-Conductor ribbon cable (color-coded):
Alpha #3581/20

* 50-pin header sockets (you'll need two):
AMP #499566-2
or Alpha #FCC-220-50

* 20-pin header socket (you'll need one):
AMP #499568-4,
or Alpha #FCC-220-20

* 20-position card edge connector (you'll need one):
AMP #499560-6,
or Alpha #FCC-170-20

* 34-position card edge connectors (you'll need two):
AMP #499560-3
or Alpha #FCC-170-34

* Drive and controller power connectors:
Shell: AMP #1-480424 (You'll need two shells)
Pins: AMP #60619-1 (You'll need eight pins)

SOUTH EAST MEDIA

New Software Additions

TEXT EDITORS

CEDRIC - A screen oriented TEXT EDITOR with availability of 'MENU' aid. Macro definitions, configurable 'permanent definable MACROS' - all standard features and the fastest 'global' functions in the west. A simple, automatic terminal config program makes this a real 'no hassel' product. Only 6K in size, leaving the average system over 165 sectors for text buffer - appx. 14,000 plus of free memory! Extra fine for programming as well as text.

Regular $129.95

* SPECIAL INTRODUCTION OFFER * FLEX $69.95

- - - - - - - - - - -

PAT - A full feature screen oriented TEXT EDITOR with all the best of "PIE (tm)". For those who swore by and loved only PIE, this is for you! All PIE features and much more! Too many features to list. And if you don't like these, change or add your own. PL-9 source furnished. "C" source available soon. Easily configured to your CRT, with special config section.

Regular FLEX $129.50

* SPECIAL INTRODUCTION OFFER * FLEX $79.95

SPECIAL PAT/JUST COMBO

PAT & JUST (w/source) FLEX $99.95

Note: JUST in "C" source available for OS-9

** See JUST advertising - S.E. MEDIA Catalog - this issue

PROGRAMMERS & USERS TOOLS

SOLVE - OS-9 Levels I and II only. A Symbolic Object/Logic Verification & Examine debugger. Including inline debugging, disassemble and assemble. SOLVE IS THE MOST COMPLETE DEBUGGER we have seen for the 6809 OS-9 series! SOLVE does it all! With a rich selection of monitor, assembler, disassembler, environmental, execution and other miscellaneous commands, SOLVE is the MOST POWERFUL tool-kit item you can own! Yet, SOLVE is simple to use! With complete documentation, a snap! Everyone who has ordered this package has raved! See review - 68 Micro Journal - December 1985. No 'blind' debugging here, full screen displays, rich and complete in information presented. Since review in 68 Micro Journal, this is our fastest mover!

Levels I & II only - OS-9 Regular $149.95

* SPECIAL INTRODUCTION OFFER * $69.95

** NOTE: Please note the special discounts (limited time) of assorted software in the S.E. MEDIA catalog in this and other issues.

Also please note the new policy on documentation of some S.E. MEDIA owned or licensed software products offered in their catalog, and repeated below.

Most all prgrams have the documentation in text disk file format. If you can print it out on your printer, the price is as shown in the catalog for the software. If you want us to print it out, please add $25.00. This is our average reproduction, abort run cost. On most all items, the savings is well worth your doing the printing. However, this is only done to help save you hard earned dollars. All other software has vendor furnished documentation included in the price. Another - "your choice" S.E. MEDIA feature!

SPECIAL

K-BASIC updates are now available. If you purchased K-BASIC prior to July 1, 1985 and wish to have your K-BASIC updated, please send $35 enclosed with your master disk to Southeast Media.

K-BASIC under OS-9 and FLEX will now compile TSC BASIC, XBASIC, and XPC Source Code Files

Telex 5106006630
(615)842-4600

SOUTH EAST MEDIA

5900 Cassandra Smith Rd.
Hixson, TN 37343
for information
call (615) 842-4601

CoCo OS-9™ FLEX™
SOFTWARE

K-BASIC now makes the multitude of TSC XBASIC Software available for use under OS-9. Transfer your favorite BASIC Programs to OS-9, compile them, Assemble them, and BINGO -- usable, multi-precision, familiar Software is running under your favorite Operating System!

K-BASIC (OS-9 or FLEX), Including the Assembler
!!! Special !!! ~~$199.00~~ $ 99.49

SAVE $100.00

!!! SPECIAL !!! Through Jan. 15, 1986 !!! SPECIAL !!!
$150 OFF Sculptor OS-9 or UniFLEX Development Pkg.

SCULPTOR

Microprocessor Developments Ltd.'s Commercial Application Generator Program provides a **FAST** Commercial Application Development tool unavailable to the OS-9 and UniFLEX User before. Develop any Commercial Application in 20% of the normal required time; gain easy updating or customizing. **PLUS**, the Application can also be run on MS-DOS and Unix machines! Sculptor handles input validation, complex calculations, and exception conditions as well as the normal collecting, displaying, reporting, and updating information in an orderly fashion. Key fields to 160 bytes; unlimited record size; file size should be held to 17 million records. Utilizes ISAM File Structure and B-tree Key files for rapid access. Input and Output communication with other programs and files plus a library of ISAM routines for use with C Programs. Run-time included w/ the Development package; a compiled Application only needs a Run-time License. Additional charge for Networked Units. Prices for Development Package/Run-time. Discounts available for purchases of 5 or more Run-time Packages.

```
OS-9 / UniFLEX --                     68000 UniFLEX --                       MS DOS --
IBM PC Zenix     --  $995.00/$175.00  Altos Zenix    -- $1595.00/$265.00            -- $595.00/$115.00
MS DOS Network --       *       **    UNIX           --      *       **      PC DOS --       *       **
```

* Full Developement Package ** Run Time Package Only Full OEM and Dealer Discounts Available!

!!! Special Buy Out !!! **SPECIAL** - Limited Quantity !!! Special Buy Out !!!

6809 FLEX SOFTWARE

TSC Flex Utilities	was $75.00	Now only $50.00
TSC Basic	was $75.00	Now only $35.00
TSC Text Processor	was $75.00	Now only $50.00
TSC Flex Precompiler	was $50.00	Now only $35.00
TSC Flex Basic	was $75.00	Now only $50.00
TSC Flex Diagnostic	was $75.00	Now only $50.00
TSC Text Processor	was $75.00	Now only $50.00
TSC Assembler	was $50.00	Now only $35.00
TSC Precompiler	was $50.00	Now only $35.00
TSC Editor	was $50.00	Now only $35.00
TSC Utilities	was $75.00	Now only $50.00

DISKS ONLY

TSC Extended Precompiler	was $50.00	Now only $35.00
TSC Basic Plus	was $75.00	Now only $50.00
TSC Diagnostics	was $75.00	Now only $50.00
TSC Utilities	was $75.00	Now only $50.00

MANUALS ONLY

TSC Basic Precompiler	was $25.00	Now only $20.00
TSC Text Editor	was $25.00	Now only $20.00
TSC Debug	was $25.00	Now only $20.00
TSC Mnemonic Assembler	was $25.00	Now only $20.00

COMPLETE 6800 SOFTWARE

TSC Text Processor	was $50.00	Now only $35.00
TSC Precompiler	was $50.00	Now only $35.00
TSC Diagnostics	was $75.00	Now only $50.00

DISKS ONLY - 6800 Software

TSC Editor	was $50.00	Now only $35.00
TSC Utilities	was $100.00	Now only $70.00
TSC Assemblers	was $50.00	Now only $35.00

MANUALS ONLY - 6800 Software

TSC Diagnostics	was $25.00	Now only $20.00
TSC Debug	was $25.00	Now only $20.00
TSC Text Processor	was $25.00	Now only $20.00

!!! Please Specify Your Operating System & Disk Size !!!

(615)842-4600 Telex 5106006630

SOUTH EAST MEDIA

5900 Cassandra Smith Rd.
Hixson, TN 37343
for information
call (615) 842-4601

CoCo OS-9™ FLEX™
SOFTWARE

PROGRAMMING

ASSEMBLERS

ASTRUK09 from **Southeast Media** -- A "Structured Assembler for the 6809" which requires the TSC **Macro Assembler**. F, CCF - $99.95

Macro Assembler for **TSC** -- The FLEX **STANDARD** Assembler.
Special -- CCF $35.00; F $50.00

OSM Extended 6809 Macro Assembler from **Lloyd I/O**. -- Provides local labels, Motorola S-records, and Intel Hex records; XREF. Generate OS-9 Memory modules under FLEX. FLEX, CCF, OS-9 $99.00

Relocating Assembler w/Linking Loader from TSC. -- Use with many of the C and Pascal Compilers. F,CCF $150.00

MACE, by Graham Trott from **Windrush Micro Systems** -- Co-Resident Editor and Assembler; fast interactive A.L. Programming for small to medium-sized Programs. F,CCF - $75.00

XMACE -- MACE w/ Cross Assembler for 6800/1/2/3/8 F,CCF - $98.00

TRUE CROSS ASSEMBLERS from **Computer Systems Consultants** -- Supports 1802/5, Z-80, 6800/1/2/3/8/11/HC11, 6804, 6805/HC05/146805, 6809/00/01, 6502 family, 8080/5, 8020/1/2/35/C35/39/40/48/C48/49/C49/50/8748/49, 8031/51/8751, and 68000 Systems. Assembler and Listing formats same as target CPU's format. Produces machine independent Motorola S-Text.
FLEX, CCF, OS-9, UniFLEX each - $50.00
any 3 - $100.00
the complete set w/ C Source (except the 68000 Source) - $200.00

XASM Cross Assemblers for FLEX from **Compusense Ltd**. -- This set of 6800/1/2/3/5/8, 6301, 6502, 8080/5, and Z80 Cross Assemblers uses the familiar TSC **Macro Assembler** Command Line and Soruce Code format, Assembler options, etc., in providing code for the target CPU's. Complete set, FLEX only - $150.00

CRASMB from **Lloyd I/O** -- 8-Bit Macro Cross Assembler with same features as OSM; cross-assemble to 6800/1/2/3/4/5/8/9/11, 6502, 1802, 8048 Sers, 80/85, Z-8, Z-80, TMS-7000 sers. Supports the target chip's standard mnemonics and addressing modes.
FLEX, CCF, OS-9 Full package -- $399.00

CRASMB 16.32 from **Lloyd I/O** -- Cross Assembler for the 68000.
FLEX, CCF, OS-9 $249.00

DISASSEMBLERS

SUPER SLEUTH from **Computer Systems Consultants** -- Interactive Disassembler; extremely POWERFUL! Disk File Binary/ASCII Examine/Change, Absolute or FULL Disassembly. XREF Generator, Label "Name Changer", and Files of "Standard Label Names" for different Operating Systems

Color Computer	**SS-50 Bus (all w/ A.L. Source)**
CCO (32K Req'd) Obj. Only $49.00	F, $99.00
CCF, Obj. Only $50.00	U, $100.00
CCF, w/Source $99.00	O, $101.00
CCO, Obj. Only $50.00.	

DYNAMITE + from **Computer Systems Center** -- Excellent standard "Batch Mode" Disassembler. Includes XREF Generator and "Standard label" Files. Special OS-9 options w/ OS-9 Version.

CCF, Obj. Only $100.00	CCO, Obj. Only $ 59.95
F, " " $100.00	O, " " $150.00
	U, " " $300.00

PROGRAMMING LANGUAGES

PL/9 from **Windrush Micro Systems** -- By Graham Trott. A combination Editor/Compiler/Debugger. Direct source-to-object compilation delivering fast, compact, re-entrant, ROM-able, PIC. 8 & 16-bit Integers & 6-digit Real numbers for all real-world problems. Direct control over ALL System resources, including interrupts. Comprehensive library support; simple Machine Code interface; step-by-step tracer for instant debugging. **500+ page Manual** with tutorial guide. F, CCF - **$198.00**

WHIMSICAL from **Whimsical Developments** -- Now supports **Real Numbers**. "Structured Programming" WITHOUT losing the Speed and Control of Assembly Language! Single-pass Compiler features unified, user-defined I/O; produces ROMable Code; Procedures and Modules (including pre-compiled Modules); many "Types" up to 32 bit Integers, 6-digit Real Numbers, unlimited sized Arrays (vectors only); Interrupt handling; long Variable Names; Variable Initialization; Include directive; Conditional compiling; direct Code insertion; control of the Stack Pointer; etc. Run-Time subroutines inserted as called during compilation. Normally produces 10% less code than PL/9. F and CCF - $195.00

C Compiler from **Windrush Micro Systems** by James McCosh. Full C for FLEX except bit-fields, including an Assembler. Requires the TSC Relocating Assembler if user desires to implement his own Libraries. F and CCF - **$295.00**

C Compiler from **Introl** -- Full C except Doubles and Bit Fields, streamlined for the 6809. Reliable Compiler; FAST, efficient Code. More UNIX Compatible than most.
FLEX, **CCF**, **OS-9 (Level II ONLY)**, U - $575.00

PASCAL Compiler from **Lucidata** -- ISO Based P-Code Compiler. Designed especially for Microcomputer Systems. Allows linkage to Assembler Code for maximum flexibility.
F and CCF 5" - $190.00 F 8" - $205.00

PASCAL Compiler from **Omegasoft** (now **Certified Software**) -- For the **PROFESSIONAL**; ISO Based, Native Code Compiler. Primarily for Real-Time and Process Control applications. Powerful; Flexible. Requires a "Motorola Compatible" Relo. Asmb. and Linking Loader.
F and CCF - $425.00 One Year Maint. - $100.00

K-BASIC from **LLOYD I/O** -- A "Native Code" **BASIC Compiler** which is now **Fully TSC XBASIC** compatible. The compiler compiles to Assembly Language Source Code. A NEW, **streamlined**, Assembler is now included allowing the assembly of LARGE Compiled K-BASIC Programs. Conditional assembly reduces Run-time package.
FLEX, CCF, OS-9 Compiler with Assembler - $199.00

CRUNCH COBOL from **Compusense Ltd**. -- Supports large subset of ANSII Level 1 COBOL with many of the useful Level 2 features. Full FLEX File Structures, including Random Files and the ability to process Keyed Files. Segment and link large programs at runtime, or implemented as a set of overlays. The System requires 56K and CAN be run with a single Disk System.
FLEX, CCF; Normally $199.00
Special Introductory Price (while in effect) -- $99.95

FORTH from **Stearns Electronics** -- A CoCo **FORTH** Programming Language. **Tailored to the CoCo!** Supplied on Tape, transferable to **disk**. Written in FAST ML. Many CoCo functions (Graphics, Sound, etc.). Includes an Editor, **Trace**, etc. Provides CPU Carry Flag accessibility, Fast Task Multiplexing, Clean Interrupt Handling, etc. for the "Pro". Excellent "Learning" tool!
Color Computer **ONLY** - $58.95

VISA MasterCard

** SHIPPING **
Add 2% U.S.A.
(min. $2.50)
Add 5% Surface Foreign
10% Air Foreign

SOUTH EAST MEDIA
5900 Cassandra Smith Rd.
Hixson, TN 37343
info (615) 842-4601
CoCo OS-9™ FLEX™
SOFTWARE

Availability Legends --
F = FLEX, CCF = Color Computer FLEX
O = OS-9, CCO = Color Computer OS-9
U = UniFLEX
CCD = Color Computer Disk
CCT = Color Computer Tape

™FLEX is a trademark of Technical Systems Consultants
™OS9 is a trademark of Microware

!!! Please Specify Your Operating System & Disk Size !!!

GENERAL

SOFTWARE DEVELOPMENT

BasicO9 XRef from Southeast Media -- This **BasicO9 Cross Reference** Utility is a BasicO9 Program which will produce a "pretty printed" listing with each line numbered, followed by a complete cross referenced listing of all variables, external procedures, and line numbers called. Also includes a **Program** List Utility which outputs a fast "pretty printed" listing with line numbers. Requires BasicO9 or RunB.
O & CCO obj. only -- $39.95; w/ Source - $79.95

Lucidata PASCAL UTILITIES (Requires LUCIDATA Pascal ver 3)

XREF -- produce a Cross Reference Listing of any text; oriented to Pascal Source.

INCLUDE -- Include other Files in a Source Text, including Binary; unlimited nesting capabilities.

PROFILER -- provides an Indented, Numbered, "Structogram" of a Pascal Source Text File; view the overall structure of large programs, program integrity, etc. Supplied in Pascal Source Code; requires compilation.

F, CCF -- **EACH Utility** 5" - $40.00, 8" - $50.00

DUB from Southeast Media -- A **UniFLEX** "basic" De-Compiler. Re-Create a Source Listing from UniFLEX **Compiled basic** Programs. Works w/ ALL Versions of 6809 UniFLEX basic. U - $219.95

FULL SCREEN FORMS DISPLAY from Computer Systems Consultants -- **TSC Extended BASIC** program supports any Serial Terminal with Cursor Control or Memory-Mapped Video Displays; substantially extends the capabilities of the Program Designer by providing a table-driven method of describing and using Full Screen Displays.
F and CCF, U - $25.00, w/ Source - $50.00

DISK UTILITIES

OS-9 VDisk from Southeast Media -- For **Level I** only. Use the **Extended Memory** capability of your SWTPC or Gimix CPU card (or similar format DAT) for FAST Program Compiles, CMD execution, high speed inter-process communications (without pipe buffers), etc. - SAVE that System Memory. Virtual Disk size is variable in 4K increments up to 960K. Some Assembly Required.
-- **Level I ONLY** -- OS-9 obj. only - $79.95; w/ Source - $149.95

O-F from Southeast Media -- Written in **BASIC09** (with Source), includes: **REFORMAT**, a BASIC09 Program that reformats a chosen amount of an **OS-9** disk to **FLEX Format** so it can be used normally by FLEX; and FLEX, a BASIC09 Program that does the actual **read** or **write** function to the special **O-F Transfer Disk**; user-friendly menu driven. Read the FLEX Directory, Delete FLEX Files, Copy both directions, etc. FLEX users use the special disk just like any other FLEX disk. **SPECIAL 60 DAY OFFER** O-$39.95

COPYMULT from Southeast Media -- Copy **LARGE** Disks to several **smaller** disks. FLEX utilities allow the backup of **ANY size disk** to any SMALLER size diskettes (Hard Disk to floppies, 8" to 5", etc.) by simply inserting diskettes as requested by **COPYMULT**. No fooling with directory deletions, etc. **COPYMULT.CMD** understands normal "copy" syntax and keeps up with files copied by maintaining directories for both host and receiving disk system. Also includes **BACKUP.CMD** to download any size "random" type file; **RESTORE.CMD** to restructure copied "random" files for copying, or recopying back to the host system; and **FREELINK.CMD** as a "bonus" utility that "relinks" the free chain of floppy or hard disk, eliminating fragmentation.
Completely documented Assembly Language Source Files included.
ALL 4 Programs (FLEX, 8" or 5") $99.50

COPYCAT from Lucidata -- Pascal **NOT** required. Allows reading TSC Mini-FLEX, SSB DOS68, and Digital Research CP/M Disks while operating under FLEX 1.0, FLEX 2.0, or FLEX 9.0 with 6800 or 6809 Systems. COPYCAT will not perform miracles, but, between the program and the manual, you stand a good chance of accomplishing a transfer. Also includes some Utilities to help out. Programs supplied in **Modular Source Code** (Assembly Language) to help solve unusual problems.
F and CCF 5" - $50.00 F 8" - $65.00

☎ (615)842-4600 Telex 5106006630

SOUTH EAST MEDIA

5900 Cassandra Smith Rd.
Hixson, TN 37343
for information
call (615) 842-4601

CoCo **OS-9™ FLEX™**

SOFTWARE

FLEX DISK UTILITIES from Computer Systems Consultants -- **Eight (8)** different Assembly Language (w/ Source Code) FLEX Utilities for every FLEX Users Toolbox: **Copy** a File with **CRC Errors**; **Test Disk** for errors; **Compare** two **Disks**; a fast **Disk Backup** Program; **Edit Disk Sectors**; **Linearize Free-Chain** on the Disk; print **Disk Identification**; and **Sort** and **Replace** the Disk Directory (in sorted order). -- **PLUS** -- Ten XBASIC Programs including: A **BASIC Resequencer** with **EXTRAs** over "**RENUM**" like check for **missing** label definitions, processes **Disk** to **Disk** instead of in **Memory**, etc. Other programs **Compare**, **Merge**, or **Generate Updates** between two BASIC Programs, check BASIC **Sequence Numbers**, compare two **unsequenced** files, and 5 Programs for establishing a **Master Directory** of several Disks, and sorting, selecting, updating, and printing paginated listings of these files. A BASIC **Cross-Reference** Program, written in Assembly Language, which provides an X-Ref Listing of the Variables and Reserved Words in **TSC BASIC**, **XBASIC**, and **PRECOMPILER** BASIC Programs.
ALL Utilities **include Source** (either BASIC or A.L. Source Code).
F and CCF - $50.00
BASIC Utilities ONLY for UniFLEX -- $30.00

COMMUNICATIONS

CMODEM Telecommunications Program from Computer Systems Consultants, Inc. -- Menu-Driven; supports Dumb-Terminal Mode, Upload and Download in non-protocol mode, and the CP/M "Modem7" Christensen protocol mode to enable communication capabilities for almost any requirement. Written in "C".
FLEX, CCF, OS-9, UniFLEX: with complete Source - $100.00
without Source - $50.00

XDATA from Southeast Media -- A **COMMUNICATION** Package for the **UniFLEX** Operating System. Use with CP/M, Main Frames, other UniFLEX Systems, etc. Verifies Transmission using checksum or CRC; Re-Transmits bad blocks, etc. U - $299.99

GAME

RAPIER - 6809 Chess Program from **Southeast Media** -- Requires FLEX and Displays on **Any** Type Terminal. Features: Four levels of play. Swap side. Point scoring system. Two display boards. Change skill level. Solve Checkmate problems in 1-2-3-4 moves. Make move and swap sides. Play white or black. This is one of the **strongest** CHESS programs running on any microcomputer, estimated USCF Rating 1600+ (better than most 'club' players at higher levels). F and CCF - $79.95

VISA MasterCard

** SHIPPING **
Add 2% U.S.A.
(min. $2.50)
Add 5% Surface Foreign
10% Air Foreign

*FLEX is a trademark of Technical Systems Consultants
**OS9 is a trademark of Microware

SOUTH EAST MEDIA
5900 Cassandra Smith Rd.
Hixson, TN 37343
info (615) 842-4601
CoCo OS-9™ FLEX™
SOFTWARE

Availability Legends --
F = FLEX, CCF = Color Computer FLEX
O = OS-9, CCO = Color Computer OS-9
U = UniFLEX
CCD = Color Computer Disk
CCT = Color Computer Tape

!!! Please Specify Your Operating System & Disk Size !!!

Telex 5106006630

(615)842-4600

SOUTH EAST MEDIA

5900 Cassandra Smith Rd.
Hixson, TN 37343
for information
call (615) 842-4601

CoCo OS-9™ FLEX™
SOFTWARE

BUSINESS

WORD PROCESSING

SCREDITOR III from **Windrush Micro Systems** -- Powerful Screen-Oriented Editor/Word Processor. Almost 50 different commands; over 300 pages of Documentation with Tutorial. Features Multi-Column display and editing, "decimal align" columns (AND add them up automatically), multiple keystroke macros, even/odd page headers and footers, imbedded printer control codes, all justifications, "help" support, store common command series on disk, etc. Use supplied "set-ups", or remap the keyboard to your needs. Except for proportional printing, this package will DO IT ALL!

6800 or 6809 FLEX or SSB DOS, OS-9 - $175.00

STYLO-GRAPH from **Great Plains Computer Co.** -- A full-screen oriented WORD PROCESSOR -- (uses the 51 x 24 Display Screens on CoCo FLEX/STAR-DOS, or PBJ Wordpak). Full screen display and editing; supports the Daisy Wheel proportional printers.
NEW PRICES ---> CCF and CCO - $99.95, F or O - $179.95, U - $299.95

STYLO-SPELL from **Great Plains Computer Co.** -- Fast Computer Dictionary. Complements Stylograph.
NEW PRICES ---> CCF and CCO - $69.95, F or O - $99.95, U - $149.95

STYLO-MERGE from **Great Plains Computer Co.** -- Merge Mailing List to "Form" Letters, Print multiple Files, etc., through Stylo.
NEW PRICES ---> CCF and CCO - $59.95, F or O - $79.95, U - $129.95

JUST from **Southeast Media** -- Text Formatter developed by Ron Anderson; for Dot Matrix Printers, provides many unique features. Output "Formatted" Text to the Display. Use the PPRINT.CMD supplied for producing multiple copies of the "Formatted" Text on the Printer **INCLUDING IMBEDDED PRINTER COMMANDS** (very useful at other times also, and worth the price of the program by itself). "User Configurable" for adapting to other Printers (comes set up for Epson MX-80 with Graftrax); up to ten (10) imbedded "Printer Control Commands". Compensates for a "Double Width" printed line. Includes the normal line width, margin, indent, paragraph, space, vertical skip lines, page length, page numbering, centering, fill, justification, etc. Use with PAT or any other editor.

* Now supplied as a two disk set:
Disk #1: JUST2.CMD object file, JUST2.TXT PL9 source: FLEX - CC
Disk #2: JUSTSC object and source in C: FLEX - OS9 - CC

The JTSC and regular JUST C source are two separate programs. JTSC compiles to a version that expects TSC Word Processor type commands, (.pp .sp .ce etc.) Great for your older text files.

The C source compiles to a standard syntax JUST.CMD object file. Using JUST syntax (.p .u .y etc.) with all JUST functions plus several additional printer formatting functions. Reference the JUSTSC C source. For those wanting an excellent **BUDGET PRICED** word processor, with features none of the others have. This is it!

Disk (1) - PL9 FLEX Version only - F & CCF - $49.95
Disk Set (2) - F & CCF & OS9 (C version) - $69.95

SPELLB "Computer Dictionary" from **Southeast Media** -- **OVER 120,000 words! Look up a word** from within your Editor or Word Processor (with the SPH.CMD Utility which operates in the FLEX UCS). Or check and update the Text after entry; ADD WORDS to the Dictionary, "Flag" questionable words in the Text, "View a word in context" before changing or ignoring, etc. **SPELLB** first checks a "Common Word Dictionary", then the normal Dictionary, then a "Personal Word List", and finally, any "Special Word List" you may have specified. **SPELLB** also allows the use of **Small Disk Storage** systems.
!! SPECIAL LIMITED TIME OFFER !! F and CCF - $99.95

DATA BASE - ACCOUNTING

XDMS from **Westchester Applied Business Systems** -- Powerful DBMS; M.L. program will work on a **single sided 5" disk**, yet is F-A-S-T. Supports Relational, Sequential, Hierarchical, and Random Access File Structures; has Virtual Memory capabilities for **Giant** Data Bases. **XDMS Level I** provides an "entry level" System for defining a Data Base, entering and changing the Data, and producing Reports. **XDMS Level II** adds the POWERFUL "GENERATE" facility with an English Language Command Structure for manipulating the Data to create new File Structures, Sort, Select, Calculate, etc. **XDMS Level III** adds special "Utilities" which provide additional ease in setting up a Data Base, such as copying old data into new Data Structures, changing System Parameters, etc.

XDMS System **Manual** - $24.95
XDMS Lvl I - F & CCF - $129.95
XDMS Lvl II - F & CCF - $199.95
XDMS Lvl III - F & CCF - $269.95

ACCOUNTING PACKAGES -- **Great Plains Computer Co.** and **Universal Data Research, Inc.** both have Data Base and Business Packages written in TSC XBASIC for FLEX, CoCo FLEX, and UniFLEX.

MISCELLANEOUS

TABULA RASA SPREADSHEET from **Computer Systems Consultants** -- TABULA RASA is similar to DESKTOP/PLAN; provides use of tabular computation schemes used for analysis of business, sales, and economic conditions. Menu-driven; extensive report-generation capabilities. Requires **TSC's Extended BASIC.**

F and CCF, U - $50.00, w/ Source - $100.00

DYNACALC from **Computer Systems Center** -- Electronic Spread Sheet for the 6809.

F and SPECIAL CCF - $200.00, U - $395.00

FULL SCREEN INVENTORY/MRP from **Computer Systems Consultants** -- Use the Full Screen Inventory System/Materials Requirement Planning for maintaining inventories. Keeps item field file in alphabetical order for easier inquiry. Locate and/or print records matching partial or complete item, description, vendor, or attributes; find backorder or below stock levels. Print-outs in item or vendor order. MRP capability for the maintenance and analysis of Hierarchical assemblies of items in the inventory file. Requires **TSC's Extended BASIC.**

F and CCF, U - $50.00, w/ Source - $100.00

FULL SCREEN MAILING LIST from **Computer Systems Consultants** -- The Full Screen Mailing List System provides a means of maintaining simple mailing lists. Locate all records matching on partial or complete name, city, state, zip, or attributes for Listings or Labels, etc. Requires **TSC's Extended BASIC.**

F and CCF, U - $50.00, w/ Source - $100.00

DIET-TRAC Forecaster from **Southeast Media** -- An XBASIC program that plans a diet in terms of either calories and percentage of carbohydrates, proteins and fats (C P G%) or grams of Carbohydrate, Protein and Fat food exchanges of each of the six basic food groups (vegetable, bread, meat, skim milk, fruit and fat) for a specific individual. Sex, Age, Height, Present Weight, Frame Size, Activity Level and Basal Metabolic Rate for normal individual are taken into account. Ideal weight and sustaining calories for any weight of the above individual are calculated. Provides number of days and daily calendar after weight goal and calorie plan is determined.

F - $59.95, U - $89.95

VISA MasterCard

** SHIPPING **
Add 2% U.S.A.
(min. $2.50)
Add 5% Surface Foreign
10% Air Foreign

SOUTH EAST MEDIA
5900 Cassandra Smith Rd.
Hixson, TN 37343
info (615) 842-4601
CoCo OS-9™ FLEX™
SOFTWARE

Availability Legends --
F = FLEX, CCF = Color Computer FLEX
O = OS-9, CCO = Color Computer OS-9
U = UniFLEX
CCD = Color Computer Disk
CCT = Color Computer Tape

™FLEX is a trademark of Technical Systems Consultants
™OS9 is a trademark of Microware

!!! Please Specify Your Operating System & Disk Size !!!

A RADIX SORT IMPLEMENTATION ON THE M68000

Steve Franck
Principal Software Engineer
Motorola, Inc.

Many applications for computers involve the sorting of large amounts of data. This can be a very time-consuming operation, and as a result much study has been done to find ways to sort data more quickly. Recently scientists have been applying arrays of computing elements to the problem in an effort to gain speed through parallelism, but for most of us the available hardware consists of simple Von Neumann uniprocessors. Even so, a good sorting algorithm and an efficient implementation on a suitable processor can make quite a difference.

PROS/CONS OF THE RADIX SORT

There are many environments in which the radix sort is an excellent choice for the algorithm. It has in its favor these attributes:

— A single radix sort routine, judiciously programmed, can be used to sort on keys any number of bytes in length—that is, two different implementations of the radix sort algorithm are not required if one file to be sorted has 5-byte keys and another has 12-byte keys.

— This algorithm is absolutely insensitive to the pre-sorted arrangement of the data. Some sorts perform significantly worse if the input data are already sorted, or reverse sorted.

— The radix sort is almost completely insensitive to the distribution of keys within the data. Some sorts perform significantly worse if the input keys are clustered in the collating sequence.

— Radix is a stable sort, meaning that two records with the same key will retain their original order when the sort is complete. This is absolutely essential if successive sorts are to be done on a single set of data, as would be necessary to print an inventory list by vendor name, and by part number within each vendor. Although stability is a very common requirement in sorting applications, some well-known sorts, such as the shell sort, are unfortunately not stable.

— The time consumed by the radix sort is SIMPLY PROPORTIONAL to the number of records to be sorted. That is, if sorting N records takes T seconds, then sorting (100 x N) records takes about (100 x T) seconds. For most sorting algorithms the time consumed increases at a greater than linear rate (typically proportional to N x logN); hence the radix algorithm can sort a large number of records much more quickly than the other algorithms can.

There are situations for which the radix sort is not ideal. A small number of records, say 50, might be more quickly sorted using another algorithm (such as the quicksort) because of the fixed overhead of the radix sort, although the total sort time in such a case would be miniscule for virtually any algorithm. Furthermore, the radix sort requires extra memory for tables and pointers, while some other sorts require no more than the data records themselves. The memory required by the radix sort is not unusually high—2048 + (4 x N) bytes for this implementation—but in a cost-sensitive controller environment even that figure might be unbearable. Such special conditions are the exception rather than the rule, however; the radix sort is a good general-purpose sort, and very fast.

One application for which the radix sort is an excellent candidate is in business data processing, where there are often thousands of inventory items, parcels of land, student grades, etc. to sort on various keys of varying lengths. Business systems generally have enough memory that the overhead of the radix sort is not a problem. The 68000 is a popular choice for multi-user business systems because of its high performance and its large linear address space of 16 Mbytes. That address space is particularly useful in sorting; many thousands of records cannot be efficiently sorted by processors which can access directly only 64 Kbytes of memory.

THE RADIX SORT ALGORITHM

The inner workings of the radix sort are easily understood by considering the process used to sort cards with a card sorter. Picture a card sorter, with its hopper to hold the cards to be sorted, some kind of switch by which the column on which to sort can be selected, and a group of bins to hold the sorted cards. The card sorter might be more accurately called a card distributor, for all it really does is examine each card and place it in one of the bins--specifically, the bin whose value matches that of the character found on the card in the sort column.

Figure 1 shows a small card sorter. It accepts 4-character cards as input and can only sort on fields containing the digits 0 to 4 (this was done to make formatting the article easier--the algorithm works for arbitrary data). Suppose the requirement is to sort a stack of six such cards in ascending order based on the contents of the numeric field in columns 2 to 4. The cards are put face down (so that the first card in the stack is on the bottom) in the hopper and the sort column is set to column 4.

Pressing a button causes the sorter to begin sorting. The bottom card is taken from the hopper and put in bin 4 since the card has a 4 in column 4 (the sort column). Now things look like figure 2.

Eventually all the cards in the hopper have been distributed into their appropriate bins (figure 3). Notice that some bins have more than one card, and some have none at all. Where a bin has more than one card in it, the cards in the bin are in the same order as they were in the hopper.

Next collect the cards from the various bins into a pile, putting those from bin 0 first, followed by the those from bin 1, etc. This new stack is sorted in ascending order on column 4. This new pile goes back into the hopper and the sort column is advanced to column 3 (figure 4).

Again, pressing a button starts the sort, this time on column 3. Things look like figure 5 when this second pass is done.
Now collect the cards again, bin 0 first, then bin 1, etc. This new stack, which is sorted in ascending order on columns 3 and 4, goes back into the hopper. The sort column is advanced to column 2 (see figure 6).

Pressing the button distributes the cards a third time, so they appear as shown in figure 7.

Collecting the cards a final time yields the following stack, which is sorted on the field in columns 2 to 4:

```
D430
F332
B314
E114
A114
C043
```

Notice that "A114" and "E114" are still in their original order, so the sort was stable.

If the requirement had instead been for a descending sort, the only difference in the procedure would have been to always collect the bins in reverse order. That is, the cards in bin 4 get put at the front, followed by the cards from bin 3, etc.

IMPLEMENTATION DETAILS

Coded in the 68000 structured assembly language, this implementation of the radix sort is quite fast. On an 8 Mhz 68000 executing with no wait states it can sort 10,000 records on a 4-byte key in just over half a second. Only 13 seconds are required to sort 100,000 records on a 10-byte key. This performance is due in part to the 68000's large set of general, 32-bit registers and its advanced addressing modes, but the linear relationship between number of records and sorting time which characterizes the radix sort is largely responsible.

The radix used is 256, meaning that in each pass a single byte of the key of each record is examined and interpreted as an unsigned integer in the range 0 to 255. This simple decision has interesting consequences:

— Since the routine is prepared to see any value which can be represented by an 8-bit byte, the routine can sort ANYTHING. Key bytes need not be numeric digits, or even ASCII.

— Since a byte is processed in each pass, keys of different lengths can be processed simply by making the appropriate number of passes. That is, a 5-byte key requires 5 passes, while a 12-byte key requires 12 passes.

The card hopper is represented as a linked list of records. A record is simply a sequence of bytes in memory, starting at an even address, of which the first four bytes are reserved for use by the radix sort routine. Byte five is the first byte of what the caller of the sort routine actually considers to be data. Figure 8 shows the six cards used in the example, depicted as a linked list of records as required by the routine.

The record at 00 01 20 4C is the first record in the list. The first four bytes contain the address of the second record in the list, 00 02 41 B0. The remaining bytes represent the data of the first card, which have been arbitrarily represented in ASCII. In ASCII, hex 41 31 31 34 represents "A114". Notice that the first four bytes of the sixth record are all zero; a zero in the "pointer to next record" field has been reserved to mean that there are no more records, that this one is the end of the list.

Each bin is in fact a queue--that is, a linked list of records for which a head and tail pointer are maintained. Since a radix of 256 requires 256 bins, 256 queue headers are needed to keep track of the distributed records. For efficiency reasons these headers are broken into two tables. The first is a table of 256 4-byte entries which point to the first (or HEAD) items in the various queues. Entry 0 points to the first record in queue 0, etc. The second table is also a table of 256 4-byte entries, but entry 0 of this table points to the last (or TAIL) record in queue 0, etc. An empty queue has a head of 0, and its tail points to its head. The structure of the bins is depicted in figure 9.

THE 68000 ASSEMBLY CODE

Knowing what the hopper and bins actually look like, it is easy to pseudocode the radix sort. The listing at the end of this article contains pseudocode in the routine headers. The actual code is broken into three routines--the user calls RADIX_SORT directly, and it calls the others.

The 68000 has a large set of general purpose registers which make a convenient parameter passing medium for assembly language routines. This implementation of the radix sort uses registers for its interface, although changing to a Pascal or C interface is trivial.

Four registers contain arguments to the routine:

— D0 contains a flag indicating whether the sort should be in ascending or descending order.

— D1 contains the offset into the record at which the key field starts.

— D2 contains the length of the key field.

— A0 points to the first record in the linked list of records to be sorted.

For example, to sort a linked list of records in ascending order on the contents of bytes 11 through 18,

— Set D0 to 0 to select an ascending sort.

— Set D1 to 11.

— Set D1 to 8 (there are 8 bytes from byte 11 to byte 18).

— Set A0 to the address of the first record in the list.

— Call the RADIX_SORT subroutine.

On return from the subroutine, all registers will be the same as they were except for A0, which points to the first record in the new, sorted list.

Extensive use was made of the structured syntax and macro capabilities of the Motorola 68000 assembler to enhance the readability of the code. However, not all 68000 assemblers will have these features, so an understanding of the actual code produced is necessary.

The SAVE and RESTORE macros use the MOVEM.L instruction to push and pop groups of registers to and from the stack. The meaning of the FOR macro is straightforward: for example

```
 FOR D5 = #FIRST TO #LAST DO
        .
        .
        .
 ENDF
```

produces code that looks something like this:

```
      MOVE  #FIRST,D5
LOOP  CMP   #LAST,D5
      BHI   OUT
```

```
        ADD    #1,D5
        BRA    LOOP
OUT
```

Similarly,

```
        REPEAT
          .
          .
          .
        UNTIL <EQ>
```

produces code on this order:

```
LOOP
          .
          .
          .
        BNE    LOOP
```

The most interesting section of code is the ten instruction loop in the internal subroutine SORT_INTO_BINS, since this loop is executed once for every character of the sort field in every record to be sorted. It is the only portion of code which affects the performance of the sort on large numbers of records.

The 13 microsecond performance of this loop can be attributed largely to the ease with which the 68000 handles 32-bit data, and to its linear 16 Mbyte address space. A 16-bit processor can only access more than 64 Kbytes of memory by constantly changing segmentation registers; this would significantly decrease the performance of the sort.

THE NEW 68020

As fast as the 68000 is, Motorola's new 32-bit processor called the 68020 is far faster. While fully upward compatible with user mode software written for the 68000, the 68020 is a substantially improved design. The following enhancements contribute to the impressive speedup in execution of the radix sort algorithm:

— The standard clock rate of the 68020 is 16.67 Mhz, as opposed to 8 Mhz for the 68000. Assuming you could construct a no-wait-state memory for the 68020, the clock rate alone would cut execution time in half.

— The 68020 has a 32-bit data bus, twice the width of the 68000 data bus. The critical loop does four data accesses, three of which are for 32-bit entities. The 68000 requires two memory cycles to read or write such a datum, while the 68020 only requires one.

— The 68020 adds new instructions and addressing modes which allow the critical loop to be rewritten from ten instructions to six.

— The 68020 can access memory in three processor clock cycles, while it takes the 68000 four.

— The 68020 has a built-in instruction cache. The tiny critical loop would be cached immediately and thereafter could be fetched at a rate of 2 processor clock cycles per 32 bits.

— The 68020's execution unit is decoupled from its interface to memory. Therefore the chip does not wait for a write to memory to complete before starting to execute the next instruction.

Just for reference, the meat of the code for SORT_INTO_BINS looks like this for the 68020, with only six instructions in the critical loop:

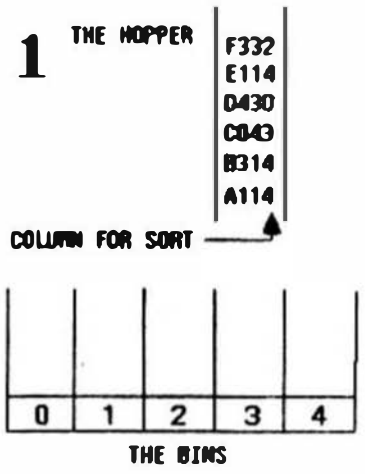
```
        CLR   D0
        BRA   TEST

        REPEAT

          MOVEB  (A0,D4.L),D0
          MOVEL  A0,([A6,D0*4])
          MOVEL  A0,(A6,D0*4)
          MOVEL  (A0),A0
TEST      TST.L    A0

        UNTIL <EQ>
```

Assuming no-wait-state memory and longword-aligned data, the critical loop would take 36 cycles. That amounts to less than 2.2 usec on a 16.67 Mhz 68020, fully six times as fast as an 8 Mhz 68000! Remember that wait states on memory would add to the execution time, but the on-chip cache would render accesses to memory for program code unnecessary—therefore wait states would only affect the four data cycles.

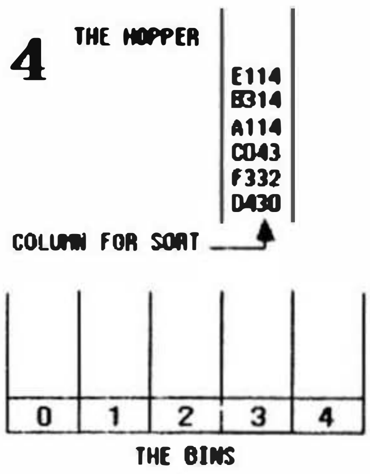
SUMMARY

The choice of an appropriate algorithm and the selection of hardware with the necessary architectural features for efficient execution of the algorithm are both essential in achieving high performance. If your application demands fast, stable sorts of large amounts of data, then the radix sort algorithm—and this implementation of it on the 68000 microprocessor family—should be of great interest.

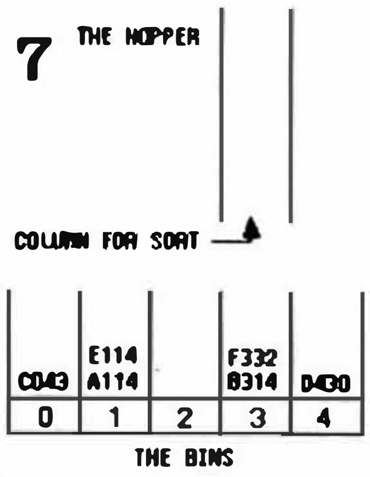
1



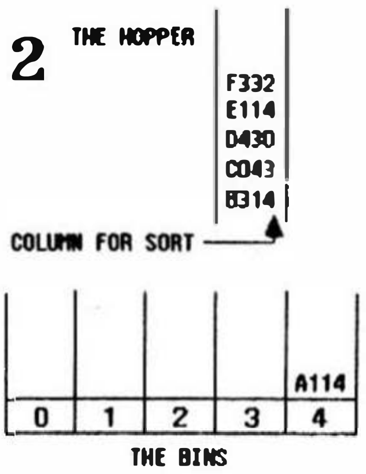
4



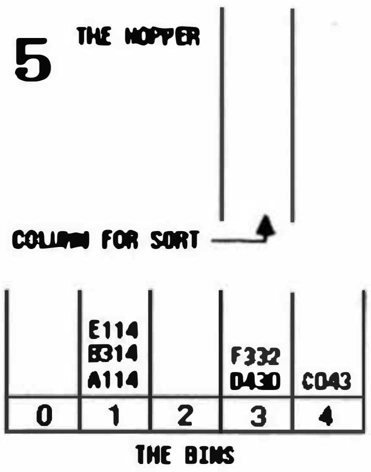
7



2



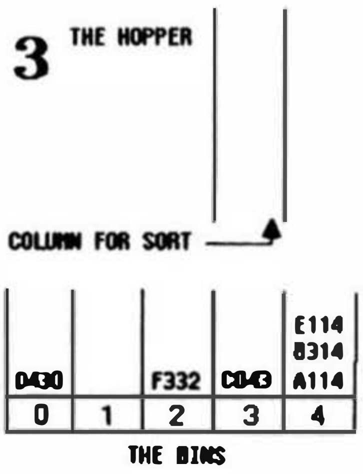
5



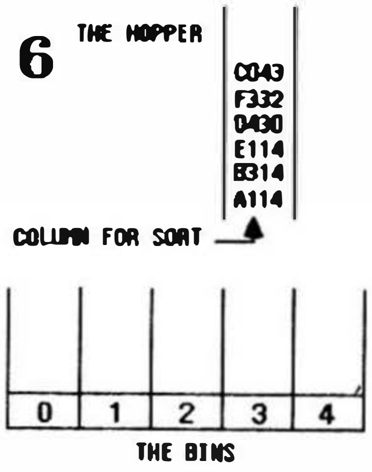
8

ADDRESS	CONTENTS OF MEMORY
00 01 20 4C	00 02 41 B0 41 31 31 34
00 02 41 B0	00 01 30 02 42 33 31 34
00 01 30 02	00 01 FE 24 43 30 34 33
00 01 FE 24	00 01 A2 2C 44 34 33 30
00 01 A2 2C	00 02 01 32 45 31 31 34
00 02 01 32	00 00 00 00 46 33 33 32

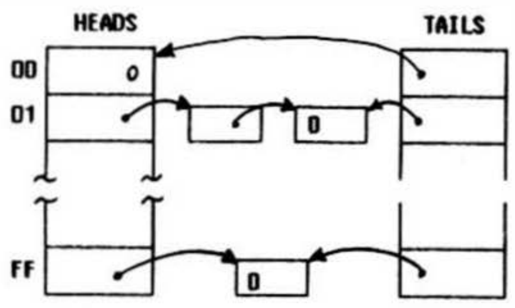
3



6



9



```
                              ***************************************************************************
                              ***                                                                     ***
                              ***    EXTERNAL SUBROUTINE:  RADIX_SORT -- General purpose RADIX sort.  ***
                              ***                                                                     ***
                              ***    DESCRIPTION:  Sorts a linked list of records according to a key  ***
                              ***        contained within the record.  The sort may be ascending or   ***
                              ***        descending.  The key may be 0 to 65535 bytes long and may be ***
                              ***        located anywhere within the record (after the link field).   ***
                              ***                                                                     ***
                              ***    NOTES:  The RADIX sort is stable; that is, records with equal keys ***
                              ***        will retain their original order relative to one another.   ***
                              ***                                                                     ***
                              ***    ENTRY CONDITIONS AND CALLING SEQUENCE:                           ***
                              ***        D0.W = Options:                                              ***
                              ***               bit 15 = 0 selects ascending sort, 1 selects descending. ***
                              ***        D1.W = byte position of key within record.                   ***
                              ***        D2.W = length of key in bytes.                               ***
                              ***        A0   = pointer (address) to first record in list to be sorted. ***
                              ***        Each record must start on a word boundary on the 68000 and 68010. ***
                              ***        Each record must have as its first 4 bytes a pointer to the  ***
                              ***        next record in the list.  The last record's pointer must be 0. ***
                              ***        2144 bytes of stack must be available.                       ***
                              ***        BSR    RADIX_SORT                                            ***
                              ***                                                                     ***
                              ***    EXIT CONDITIONS DIFFERENT FROM ENTRY:                            ***
                              ***        A0   = pointer (address) to first record in sorted list.     ***
                              ***        No records will have been moved, but the pointer fields will ***
                              ***        have been changed to reflect the sorted order.               ***
                              ***                                                                     ***
                              ***    REGISTER USAGE:  (A)rgument  (D)estroyed  (P)reserved  (R)eturned ***
                              ***                                                                     ***
                              ***              0   1   2   3   4   5   6   7      SR hi    SR lo (CCR) ***
                              ***        D:    A   A   A   P   P   P   .   .      .         D          ***
                              ***        A:    AR  .   .   .   .   P   P   P                           ***
                              ***                                                                     ***
                              ***        D3 contains the offset within the record to the current byte. ***
                              ***        D4 contains the offset within the record to the most signif. byte.***
                              ***        A5 is the address of a table of head pointers for 256 'bins.' ***
                              ***        A6 is the address of a table of tail pointers for the 'bins.' ***
                              ***                                                                     ***
                              ***    ALGORITHM USED IN THIS IMPLEMENTATION:                           ***
                              ***        Allocate space for 256 'bins' (head and tail pointers for linked ***
                              ***                        lists) in which to put records.              ***
                              ***        For each byte in key field (going from least to most significant) ***
                              ***           Initialize the 256 bins to be empty.                      ***
                              ***           Sort the records by the current key byte into the various bins.***
                              ***           Collect the bins into a big linked list.                  ***
                              ***        Deallocate space for the bins and return the last big list   ***
                              ***                        generated--it is the sorted list.            ***
                              ***                                                                     ***
                              ***    INTERNAL NOTES:  This implementation uses a radix of 256, so one ***
                              ***        byte of the key is processed in each pass.                   ***
                              ***        Stack usage:               4 bytes for the PC to return to.  ***
                              ***                         7 * 4 =   28 bytes for preserving registers. ***
                              ***                       256 * 4 = 1024 bytes for head pointers.       ***
                              ***                       256 * 4 = 1024 bytes for tail pointers.       ***
                              ***                                   32 bytes for calling internal subroutines. ***
                              ***                                   32 bytes reserved for future growth. ***
                              ***                                 ------                              ***
                              ***                                  2144 bytes of stack required.      ***
                              ***                                                                     ***
                              ***************************************************************************

                                          XDEF      RADIX_SORT               Entry point for RADIX sort routine.

                 00000100     RADIX       EQU       256                      The radix for the sort.  Since we sort on a
                              *                                              byte at a time, our radix is 256.  This
                              *                                              must not be changed!  The algorithm
                              *                                              used only works for a radix of 256.

                                          OFFSET    0                        Define TABLE_SIZE, the length of a table of
      00000000   00000400                 DS.L      RADIX                    longword pointers with one entry for each
                 00000400     TABLE_SIZE  EQU       *                        value assumable within the radix.

                 00000000     FIRST_BIN   EQU       0                        Define the range of bin numbers that exist
                 000000FF     LAST_BIN    EQU       RADIX-1                  for this radix.  A bin exists for each
                              *                                              value assumable within the radix.

                 00000008                 SECTION   9
                                          OPT       BRS

    8 00000000                RADIX_SORT:

                              *
                              *  Get set up; allocate space for bins in which to put records.
                              *
    8 00000000                            SAVE      D3-D5/A5/A6              Save registers on the stack.

    8 00000004   4FEFF800                 LEA       -2*TABLE_SIZE(SP),SP     Allocate on the stack two tables.  Each
    8 00000008   4BD7                     LEA       (SP),A5                  table consists of 256 longwords, which will
    8 0000000A   4DEF0400                 LEA       TABLE_SIZE(SP),A6        be used as pointers.  A5 points to one of
                              *                                              the tables, the head pointers,
                              *                                              and A6 to the other, the tail pointers.
```

```
                                *
                                *  Calculate the offsets to the least and most significant key bytes.
                                *
8 0C00000E 4283                        CLR.L     D3                        D3.L <-- offset into record of most
8 0C000010 3601                        MOVE      D1,D3                     significant key byte (the first).
8 0C000012 4284                        CLR.L     D4                        D4.L <-- length of the key in bytes.
8 0C000014 3802                        MOVE      D2,D4                     *
8 0C000016 D883                        ADD.L     D3,D4                     D4.L <-- offset into record of least
8 0C000018 5384                        SUB.L     #1,D4                     significant key byte = offset to 1st key
                                *                                          byte + key length - 1.

                                *
                                *  For each byte in the key field, from least to most significant,
                                *
                                       FOR.L     D4 = D4 DOWNTO D3 DO

                                *
                                *     Initialize the 256 bins (linked lists) to be empty.
                                *
8 0C00001E                               SAVE      A5/A6                   Save table pointers.

                                         FOR       D5 = #FIRST_BIN TO #LAST_BIN DO For each bin,
8 0C000028 2CCD                            MOVE.L  A5,(A6)+              .  Make tail pointer point to head pointer.
8 0C00002A 429D                            CLR.L   (A5)+                 .  Clear the head pointer.
                                         ENDF                            .

8 0C000034                               RESTORE   A5/A6                   Restore table pointers.

                                *
                                *     Sort the records on the current byte.
                                *
8 0C000038 6112                          BSR       SORT_INTO_BINS          Sort the linked list (A0) into the bins
                                *                                          (A6) based on the current key byte (D4)
                                *                                          of each record.
8 0C00003A 6136                          BSR       COLLECT_BINS            Collect the bins (A5 and A6) in ascending
                                *                                          or descending order (D0) into a big
                                *                                          linked list (A0).


                                       ENDF



                                *
                                *  Deallocate space for the bins and return the last big list generated--it
                                *                         is the sorted list.
                                *
8 0C000042 4FEF0800                    LEA       2*TABLE_SIZE(SP),SP       Adjust the stack pointer to reclaim the
                                *                                          space we took for the two tables.

8 0C000046                             RESTORE   D3-D5/A5/A6               Restore registers and return.
8 0C00004A 4E75                        RTS                                 *
                                ***********************************************************************
                                ***                                                                 ***
                                ***   SUBROUTINE:  SORT_INTO_BINS -- Sort list into bins on 1 byte. ***
                                ***                                                                 ***
                                ***   DESCRIPTION:  This routine takes all the records in the big linked  ***
                                ***       list and deposits each in the bin corresponding to the value of ***
                                ***       the current byte in its key field.                          ***
                                ***                                                                 ***
                                ***   NOTES:  In the critical loop (the set of code that gets executed once ***
                                ***       for each record in the big list), the number of 68000 processor ***
                                ***       cycles for each instruction is in brackets.  An 8 Mhz 68000 with ***
                                ***       no wait states is assumed.                                  ***
                                ***                                                                 ***
                                ***       It would have been prettier to use the WHILE structured construct,***
                                ***       but we gain a little performance by coding the loop directly.  ***
                                ***                                                                 ***
                                ***       32 bytes of stack have been reserved for this routine, including ***
                                ***       the return PC.  Currently 12 bytes are used.                ***
                                ***                                                                 ***
                                ***   ENTRY CONDITIONS AND CALLING SEQUENCE:                          ***
                                ***                                                                 ***
                                ***       D4   = offset into record of current key byte.              ***
                                ***       A0   = pointer to the big list of records to be sorted into bins. ***
                                ***       A6   = pointer to table of tail pointers.                   ***
                                ***       BSR     SORT_INTO_BINS                                      ***
                                ***                                                                 ***
                                ***   EXIT CONDITIONS DIFFERENT FROM ENTRY:                           ***
                                ***                                                                 ***
                                ***       The records are no longer linked together into a single linked ***
                                ***       list pointed to by A0.  Instead, the bins defined by the tables ***
                                ***       of head and tail pointers now contain the records.          ***
                                ***       A0 is garbage (its content at entry is meaningless now anyway). ***
                                ***                                                                 ***
                                ***   REGISTER USAGE:  (A)rgument  (D)estroyed  (P)reserved  (R)eturned ***
                                ***                                                                 ***
                                ***          0   1   2   3   4   5   6   7      SR hi     SR lo (CCR) ***
                                ***     D:   P   .   .   .   A   P   .   .      .         D           ***
                                ***     A:   AD  P   .   .   .   .   A   P                            ***
                                ***                                                                 ***
                                ***       D0 is used to build an index into the tables of pointers.   ***
                                ***       A1 points to the tail record in the bin which is indexed.   ***
                                ***                                                                 ***
                                ***   ALGORITHM USED IN THIS IMPLEMENTATION:                          ***
                                ***                                                                 ***
```

```
                             ***        While the big list is not empty,                                ***
                             ***           Examine the current byte in the key field of the first record ***
                             ***                       in the big list.                                 ***
                             ***           Find the bin corresponding to the byte's value ($00 to $FF). ***
                             ***           Remove the record from the big list and put it in the bin.   ***
                             ***                                                                        ***
                             ***************************************************************************

              C000000E                SECTION  3

  8 0C000C4C                 SORT_INTO_BINS:

  8 0C000C4C                          SAVE      D0/A1                  Save registers.

                             *
                             *  While the big list is not empty,
                             *
  8 0C000C50 6016                     BRA       TEST_DONE              Start at the bottom in case list is empty.

                                      REPEAT

                             *
                             *     Examine the current byte in the key field of the first record
                             *                 in the big list.
                             *     Find the bin corresponding to the byte's value ($00 to $FF).
                             *
  8 0C000C52 4240                       CLR       D0                   [ 4]  D0.W <-- byte from this record's key.
  8 0C000C54 10304200                   MOVE.B    0(A0,D4.L),D0        [14]  *
  8 0C000C58 D040                       ADD       D0,D0                [ 4]  D0.W <-- byte * 4 (for use as an
  8 0C000C5A D040                       ADD       D0,D0                [ 4]  index into a table of longwords).
  8 0C000C5C 2276C000                   MOVE.L    0(A6,D0),A1          [18]  A1 <-- tail pointer for that bin.

                             *
                             *     Remove the record from the big list and put it in the bin.
                             *
  8 0C000C60 2288                       MOVE.L    A0,(A1)              [14]  Make the last record in that bin
                             *                                               point to this new record.
  8 0C000C62 20880C00                   MOVE.L    A0,0(A6,D0)          [20]  Make the tail pointer for that bin
                             *                                               point to this new record.
  8 0C000C66 2050                       MOVE.L    (A0),A0              [12]  Advance to the next record.
  8 0C000C68 2008            TEST_DONE  MOVE.L    A0,D0                [ 4]  Test A0 (set condition codes).

                                      UNTIL     <EQ>                   [10]  Repeat this until a 0 link is found.
                             *                                         -----
                             *                                         [104] cycles = 13 usec @ 8 Mhz


  8 0C000C6C                          RESTORE   D0/A1                  Restore registers and return.
  8 0C000C70 4E75                     RTS                              *

                             ***************************************************************************
                             ***                                                                      ***
                             ***     SUBROUTINE:  COLLECT_BINS -- Collect bins into a single linked list. ***
                             ***                                                                      ***
                             ***     DESCRIPTION:  This routine takes the records from the various bins ***
                             ***         and strings them all together into one big linked list which ***
                             ***         reflects the sorting that has been done so far.              ***
                             ***                                                                      ***
                             ***     NOTES:  32 bytes of stack have been reserved for this routine,   ***
                             ***         including the return PC.  Currently 16 bytes are used.       ***
                             ***                                                                      ***
                             ***     ENTRY CONDITIONS AND CALLING SEQUENCE:                           ***
                             ***                                                                      ***
                             ***         D0    = options as defined under RADIX_SORT.                 ***
                             ***         A5    = pointer to table of head pointers.                   ***
                             ***         A6    = pointer to table of tail pointers.                   ***
                             ***         BSR      COLLECT_BINS                                        ***
                             ***                                                                      ***
                             ***     EXIT CONDITIONS DIFFERENT FROM ENTRY:                            ***
                             ***                                                                      ***
                             ***         A0    = the big linked list containing all records.          ***
                             ***                                                                      ***
                             ***     REGISTER USAGE:   (A)rgument  (D)estroyed  (P)reserved  (R)eturned ***
                             ***                                                                      ***
                             ***               0   1   2   3   4   5   6   7      SR hi     SR lo (CCR) ***
                             ***         D:    A   .   .   .   .   .   .   .      .         D         ***
                             ***         A:    R   .   .   .   .   A   A   P                          ***
                             ***                                                                      ***
                             ***         D2 gets from each bin the pointer to the first record (head). ***
                             ***         A1 gets from each bin the pointer to the last  record (tail). ***
                             ***                                                                      ***
                             ***     ALGORITHM USED IN THIS IMPLEMENTATION:                           ***
                             ***                                                                      ***
                             ***            Initialize the big linked list to be empty                ***
                             ***            If the sort is to be ascending                            ***
                             ***               For each bin from highest ($FF) to lowest ($00)        ***
                             ***                  If the bin is not empty                             ***
                             ***                     Add the contents to the FRONT of the big linked list ***
                             ***            Else (the sort is to be descending)                       ***
                             ***               For each bin from lowest ($00) to highest ($FF)        ***
                             ***                  If the bin is not empty                             ***
                             ***                     Add the contents to the FRONT of the big linked list ***
                             ***                                                                      ***
                             ***************************************************************************
```

```
               00000008                SECTION  8

  8 00000072                     COLLECT_BINS:

  8 00000072                           SAVE      D1/D2/A1                 Save registers.
  8 00000076 91C8                      SUB.L     A0,A0                    Initialize the big linked list to be empty.

                                 *
                                 *  If the sort is to be ascending
                                 *     For each bin from highest ($FF) to lowest ($00)
                                 *        If the bin is not empty
                                 *           add the contents to the FRONT of the big linked list
                                 *
  8 00000078 4A40                      TST       D0                       If we're supposed to sort in ascending
                                       IF        <PL> THEN                order (options bit 15 = 0),

                                         FOR       D1 = #4*LAST_BIN DOWNTO #4*FIRST_BIN BY #4 DO

  8 00000082 24351000                      MOVE.L   0(A5,D1),D2          D2 <-- pointer to head record in bin.
                                           IF       <NE> THEN            If the bin isn't empty,
  8 00000088 22761000                        MOVE.L   0(A6,D1),A1        .  A1 <-- pointer to tail record in bin.
  8 0000008C 2288                            MOVE.L   A0,(A1)            .  Make this bin's last entry point to our
  8 0000008E 2042                            MOVE.L   D2,A0              .  big list and make the head pointer of our
                                           ENDI                          .  big list point to the bin's first entry.

                                         ENDF


                                 *
                                 *     Else (the sort is to be descending)
                                 *        For each bin from lowest ($00) to highest ($FF)
                                 *           If the bin is not empty
                                 *              Add the contents to the FRONT of the big linked list
                                 *
                                       ELSE                               Else (he wants it in descending order),

                                         FOR       D1 = #4*FIRST_BIN TO #4*LAST_BIN BY #4 DO

  8 000000A0 24351000                      MOVE.L   0(A5,D1),D2          D2 <-- pointer to head record in bin.
                                           IF       <NE> THEN            If the bin isn't empty, add it to front of list:
  8 000000A6 22761000                        MOVE.L   0(A6,D1),A1        .  A1 <-- pointer to tail record in bin.
  8 000000AA 2288                            MOVE.L   A0,(A1)            .  Make this bin's last entry point to our
  8 000000AC 2042                            MOVE.L   D2,A0              .  big list and make the head pointer of our
                                           ENDI                          .  big list point to the bin's first entry.

                                         ENDF

                                       ENDI


  8 000000B6                           RESTORE  D1/D2/A1                  Restore registers and return.
  8 000000BA 4E75                      RTS                                *






                                       END

****** TOTAL ERRORS       0--
****** TOTAL WARNINGS     0--
 SYMBOL TABLE LISTING


 SYMBOL NAME      SECT    VALUE          SYMBOL NAME      SECT    VALUE


 BKGRND     MACR    *                    RADIX_SQ   XDEF    8    00000000
 BSRCC      MACR    *                    RENEW      MACR    *
 BSRCS      MACR    *                    RESERVE    MACR    *
 BSREQ      MACR    *                    RESTORE    MACR    *
 BSRGE      MACR    *                    SAVE       MACR    *
 BSRGT      MACR    *                    SEL_DCR    MACR    *
 BSRHI      MACR    *                    SEL_PER    MACR    *
 BSRHS      MACR    *                    SET_BAB    MACR    *
 BSRLE      MACR    *                    SORT_INT           8    0000004C
 BSRLO      MACR    *                    TABLE_SI                00000400
 BSRLS      MACR    *                    TEST_CON           8    00000068
 BSRLT      MACR    *                    UNMASK     MACR    *
 BSRMI      MACR    *                    Z_L1.C01           8    0000001E
 BSRNE      MACR    *                    Z_L1.C03           8    00000028
 BSRPL      MACR    *                    Z_L1.C04           8    00000052
 COLLECT_           8     00000072       Z_L1.C06           8    0000009A
 ENABLE     MACR    *                    Z_L1.C09           8    00000082
 FIRST_BI                 00000000       Z_L1.C0A           8    00000090
 INHIBIT    MACR    *                    Z_L1.C0F           8    000000A0
 LAST_BIN                 000000FF       Z_L1.C10           8    000000AE
 MOVELI     MACR    *                    Z_L2.C01           8    0000003E
 POP        MACR    *                    Z_L2.C02           8    0000002E
 PUSH       MACR    *                    Z_L2.C08           8    00000092
 PUT_ADDR   MACR    *                    Z_L2.C0D           8    00000096
 PUT_BYTE   MACR    *                    Z_L2.C0E           8    000000B0
 RADIX                    00000100
```

RATBAS

RATBAS - A BASIC Preprocessor

By: Mike Johnshoy
707 Continental Circle, #1213
Mt View, CA 94040
TEL (415) 967-2048

Overview

RATBAS is a program written in TSC XBASIC which allows the use of several additional control structures when programming in TSC XBASIC. These additional control structures include repeat, repeat-until, while, break, and if-else statements spanning many lines. In addition to the new control structures, the RATBAS preprocessor allows the use of 'curly braces' - the { and } - to group statements.

Another feature of RATBAS is that it allows comments on the same line as program code. Also, an 'include' function more powerful than XBASIC's 'LIB' is now available to the programmer.

In operation the RATBAS preprocessor reads a text file containing the RATBAS program, and creates a new file which will be accepted by the TSC XBASIC preprocessor.

RATBAS Grammar and Examples

The control structures added by the RATBAS preprocessor are described by the following grammar:

```
statement ->
    repeat statement
  | repeat statement until ( condition )
  | while ( condition ) statement
  | if ( condition ) statement
  | if ( condition ) statement else statement
  | break
  | { statement list }
```

In the grammar above, 'statement' is any single legal BASIC statement in TSC Extended BASIC. The 'statement' may even be several statements on the same line separated by colons. However, a more general method of grouping statements is to use the curly braces allowed in RATBAS programs. For example, the following is a legal RATBAS program:

```
x=0 repeat print "HI" : x=x+1 until (x=10)
```

Certainly ugly code in anyone's book. The same program could be written in a more readable style by using new lines and indentation to better show the program flow. The same logic is presented below in a more readable way:

```
x=0
repeat {            !  print HI ten times
    print "HI"
    x=x+1
    } until (x=10)
```

The term (condition) in the grammar can be any legal comparison or boolean expression. Notice that the condition is not followed by a 'then', but must be enclosed in a balanced set of parenthesis.

The statement following a repeat will always be executed at least once - and if the repeat is not followed by an until (condition) it will be repeated forever. The while statement differs from the repeat in that the condition is tested before the statement is executed the first time, and the statement following the condition may never be executed at all.

The break is used in conjunction with the repeat and while statements. When a break is encountered inside a repeat or while loop, the loop is exited in the same way as if the condition had suddenly been tested and met. If repeat and/or while statements are nested at the time the break is encountered, only the deepest nested loop is broken.

The if-else construct of RATBAS is far more powerful than the if-then-else of plain vanilla BASIC. For example, look at the following code:

```
x=1
repeat {              !  count to five
    if (x=1) print "one"
```

```
     else if (x=2) print "two"
     else if (x=3) print "three"
     else if (x=4) print "four"
     else if (x=5) print "five"
     x=x+1
     } until (x>5)
```

Additional features of RATBAS include the ability to put comments on the same line as program code. Comments are indicated by a ! , with everything from the ! to the next carriage return ignored by the preprocessor.

As in TSC XBASIC, lines may be continued logically on to the next line by ending a line with a '\' character.

Better examples of all the constructs can be found in the listing of the RATBAS preprocessor itself.

Preprocessor Directives

There is only one at this point. To replace the XBASIC 'LIB' capability, RATBAS allows the programmer to type the line #include filename. The # must be the first character on a line, and indicates that the following is a preprocessor directive, not XBASIC. Included files may include other files, and be nested so long as no more than 12 files are ever open at one time.

Error Statements

The RATBAS preprocessor is capable of finding and flagging several errors. They are as follows:

illegal until - until not preceded by a
    repeat
illegal else - else not balanced by a
    preceding if
illegal right - no balancing left
    hand brace
unexpected end-of-file - end-of-file
    encounterd with unclosed RATBAS
    control structures
illegal break - break without
    repeat or while
missing double quote - unbalanced quote
missing single quote - unbalanced quote
missing left parenthesis - unbalanced
    parenthesis
missing right parenthesis - unbalanced
    parenthesis
includes nested too deep - more than 12
 files open at once

Each of these errors is printed along with the source file line number being processed when the error is detected. Note that if files have been 'included' that the line number given with the error message is the total number of lines already processed when the error is encountered.

When entering the source code for this program you will notice several additional 'compiler' errors. The capability to detect these errors was added to ease the development of the preprocessor, and they should never be encountered in normal use.

Getting RATBAS to Run

It's unfortunate, but you're going to have to type it in (unless you can find it on a bulletin board - I'll try to upload it to FLEXNET). The listing to enter is not the RATBAS source listing, but the TSC XBASIC listing. Then you can run the TSC Precompiler on that and have a working RATBAS preprocessor. If you wish to work on modifying the preprocessor itself, you should enter the RATBAS source listing of the RATBAS preprocessor and make your changes in it, recompiling the imnproved vesion with your original bootstrap version.

The Future

An entire series of RATBAS utilites is in work. These utilities include a RATBAS Beautifier, to produce nicely indented listings of ill typed code. Such a utility might help detect errors where the programmer has used indentation to show program flow that is contrary to that understood by the preprocessor. Another utilty is a combination of a program editor and verifier, which does things like counting parenthesis and curly braces. The editor portion allows a program to be 'outlined' in terms of procedures or subroutines and for the writer to move around in text by procedure name or outline level instead of just scrolling back and forth by line number. The possibilities are endless. The author is interested in hearing of any ideas or work along these lines.

RATBAS source is available from 68' Micro Journal on disk #17. See page 62.

Bit Bucket

SHRINK
by Elite Software
Box 11224
Pittsburgh PA 15238
for 16K ECB or larger systems

SHRINK is a machine language utility that will remove all unnecessary spaces from BASIC programs. It comes with a page of instructions that describe its operation and explain how to relocate the program to run on 32K systems (it is sold for 16K systems).

Removing unneeded spaces from BASIC programs will produce programs that use less memory and run faster. Unfortunately, while SHRINK does exactly what it claims to do, it is does not save significant amounts of memory or time. SHRINK would save more memory if it would delete REMarks and/or pack program lines together. For every line number that is deleted from a program 4 bytes of memory will be saved. Depending on your coding style, SHRINK may strip an average of 1 to 3 blanks per line.

Shrink does exactly what it claims to do, but I can't recommend it because it won't really save a typical BASIC programmer much time or space.

by Troy Brumley

COCOCOPY

By THE DATAMAN
420 Ferguson Avenue North
Hamilton, Ontario, Canada L8L 4Y9
$14.95 (Canadian) for 16K+ systems

COCOCOPY is bound to be a workhorse in any cassette users software library. This program will load, save, rename, or relocate any standard CoCo tape.

To use COCOCOPY just set the tape up in the recorder and CLOADM it. The program autostarts, so once it's loaded you can get right to work. A menu is displayed showing all of your options. In addition to loading, saving, renaming and relocating programs COCOCOPY will also allow you to review a tape with single key control of the AUDIO ON/OFF and MOTOR ON/OFF functions. This allows easy positioning of tapes without pulling plugs out of your recorder.

One of COCOCOPY's strong points is its ability to load a file with I/O errors. This may let you recover your only copy of an important data file or program. The file you create may not be 100% correct but at least it can be loaded for fixing.

Another strong point is the ability to relocate a machine language program. This will allow disk users to load in a program that they couldn't normally and then save it so that it will load at a new address. This is possible because COCOCOPY loads programs into a BUFFER and ignores all actual load and execute addresses. Once a program is loaded it is easy to change both the load and execute addresses. Note that this relocation will work only if the machine language program is written in Position Independent Code.

Unfortunately COCOCOPY has no provisions for direct saves to disk. This would be a nice feature for new disk users with large tape libraries. Using the present system the programs must be loaded, relocated, and saved to tape, then the new copies must be loaded in and saved in DISK BASIC.

Unlike some cassette based software, COCOCOPY would load and run with my disk controller installed. Even so, tape and disk users will need to turn their computers off and back on again to regain control of their computers. COCOCOPY is RESET PROTECTED.

COCOCOPY claims to be able to load and save some types of autostarting tapes. I don't know if this is true since I don't have any such tapes around the house besides COCOCOPY. COCOCOPY would not copy itself.

All in all COCOCOPY is a good buy for tape users. If it could save directly to disk it would be a "must have" program for disk users who buy a lot of machine language games.

by Troy Brumley

FORCE COMPUTERS

FORCE COMPUTERS, INC
727 University Ave
Los Gatos, California 95030

FORCE INTRODUCES LOWEST COST UNIX-on-VMEbus

Los Gatos, Ca., Sept. 4, 1985.......Force Computer Inc. today announced a complete, two-user VMEbus development system with standard UNIX System V. Called the microForce-1, the new high performance UNIX-on-VME system is the first very low-cost product based entirely on industry-standard VMEbus boards.

The system implements AT&T's UNIX V, Release 1.0 on the 68010 microprocessor, with 1 Mbyte of fast main memory, 1 expansion slot and 25 Mbytes of Winchester storage (22 MBytes formatted). microFORCE-1 includes two user ports, plus printer and communications ports, all delivered in a compact 16" x 16" desktop package.

According to Jim Green, Manager of Application Engineering at Force Computer, "The microFORCE-1 is primarily a software design tool. This is an excellent machine for designers who want the privacy and freedom of their own machine and who do not want to be centrally locked into a multiuser main frame. It is especially suited to OEM's interested in an inexpensive development system running UNIX for two primary uses; they can utilize it as either a stand-alone tool with direct hardware and software compatability to larger VMEbus systems or as the target system."

The microFORCE-1 provides the benefits of the advanced VMEbus architecture along with the many and varied UNIX utilities. Basically, this new machine combines the two fastest-growing standards on the market...VMEbus and UNIX, for the lowest cost and highest performance currently available.

According to Peter Schmitz, operation systems specialist at Force Computers Inc. in Los Gatos, "We've been running UNIX benchmarks on both new products, the microFORCE-1 and it's 'big brother', our six user STANDARD-30. Both products are identical under single-user benchmarks and have run as fast or faster than industry's top six benchmarked systems in many catagories."

Peter Schmitz goes on to note that the features of the microFORCE's CPU-3VA makes the machine ideal as a UNIX engine. "The MMU bypass, working with 32 Kbytes of high speed local static ram (with no-wait-states) allows rapid execution of UNIX system calls. This can be seen in the accompanying (Byte Aug. '84) benchmarks."

Features of the microFORCE-1 include:

68010 CPU, 68451 MMU and 68450 DMA.

Four serial I/O channels allow for two users, one printer and one communications port.

32 Kbyte no-wait-state SRAM allows high speed operation of the UNIX kernel.

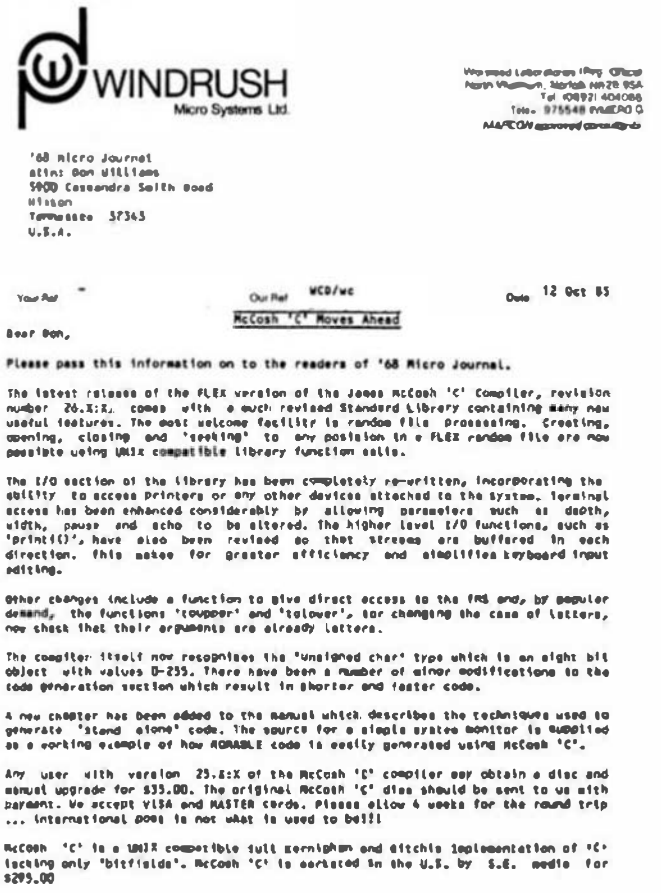
Mass memory included 5 1/4" floppy and 5 1/4" Winchester hard disk with 25 Mbytes capacity unformatted 22 Mbytes formatted.

UNIX Version 5.1 including C and Fortran 77. (with Pascal, Basic and UNIX 5.2 available soon.)

1 Mbyte DRAM on VMEbus for global memory.

Open slot for user application board, additionl memory, etc.

Rapidly growing Force Computers Inc. is now the number one independent supplier of VMEbus products and systems worldwide. The company is noted for its broad base of VMEbus products that offer customers the best price/performance ratios in the market. Force is a multinational Corporation with headquarters in Los Gatos, California and subsidiaries in Munich, West Germany and Paris, France.



WINDRUSH Micro Systems Ltd.

Westwood Laboratories (Reg. Office)
North Walsham, Norfolk NR28 9SA
Tel (0692) 404086
Telex 975548 MICRO G
MARCON approved consultants

'68 Micro Journal
attn: Don Williams
5900 Cassandra Smith Road
Hixson
Tennessee 37343
U.S.A.

Your Ref - Our Ref WCD/wc Date 12 Oct 85

McCosh 'C' Moves Ahead

Dear Don,

Please pass this information on to the readers of '68 Micro Journal.

The latest release of the FLEX version of the James McCosh 'C' Compiler, revision number 26.X:X, comes with a much revised Standard Library containing many new useful features. The most welcome facility is random file processing. Creating, opening, closing and 'seeking' to any position in a FLEX random file are now possible using UNIX compatible library function calls.

The I/O section of the library has been completely re-written, incorporating the ability to access printers or any other devices attached to the system. Terminal access has been enhanced considerably by allowing parameters such as depth, width, pause and echo to be altered. The higher level I/O functions, such as 'printf()', have also been revised so that streams are buffered in each direction. This makes for greater efficiency and simplifies keyboard input editing.

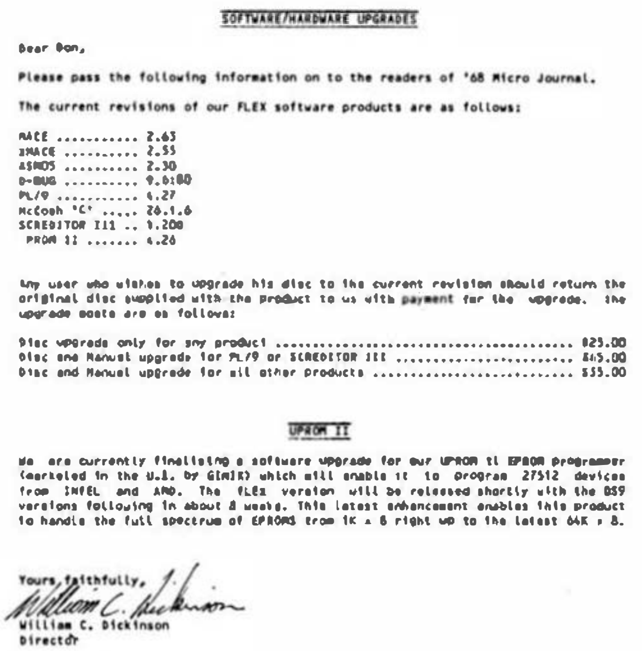
Other changes include a function to give direct access to the FMS and, by popular demand, the functions 'toupper' and 'tolower', for changing the case of letters, now check that their arguments are already letters.

The compiler itself now recognises the 'unsigned char' type which is an eight bit object with values 0-255. There have been a number of minor modifications to the code generation section which result in shorter and faster code.

A new chapter has been added to the manual which describes the techniques used to generate 'stand alone' code. The source for a simple system monitor is supplied as a working example of how ROMABLE code is easily generated using McCosh 'C'.

Any user with version 25.X:X of the McCosh 'C' compiler may obtain a disc and manual upgrade for $35.00. The original McCosh 'C' disc should be sent to us with payment. We accept VISA and MASTER cards. Please allow 4 weeks for the round trip ... international post is not what is used to be!!

McCosh 'C' is a UNIX compatible full Kernighan and Ritchie implementation of 'C' lacking only 'bitfields'. McCosh 'C' is marketed in the U.S. by S.E. Media for $295.00



SOFTWARE/HARDWARE UPGRADES

Dear Don,

Please pass the following information on to the readers of '68 Micro Journal.

The current revisions of our FLEX software products are as follows:

```
MACE ........... 2.63
XMACE .......... 2.55
ASMOS .......... 2.30
D-BUG .......... 9.6:80
PL/9 ........... 4.27
McCosh 'C' ..... 26.1.6
SCREDITOR III .. 1.200
 PROM II ....... 4.26
```

Any user who wishes to upgrade his disc to the current revision should return the original disc supplied with the product to us with payment for the upgrade. The upgrade costs are as follows:

```
Disc upgrade only for any product ................................. $25.00
Disc and Manual upgrade for PL/9 or SCREDITOR III ................. $45.00
Disc and Manual upgrade for all other products .................... $55.00
```

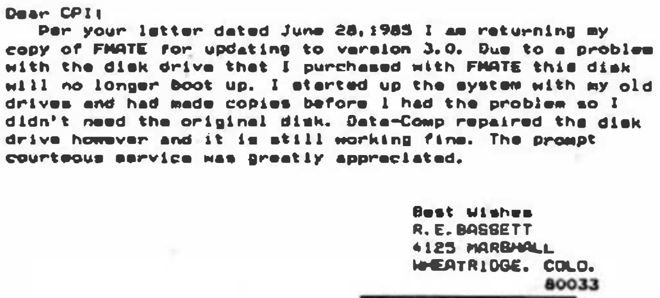
UPROM II

We are currently finalising a software upgrade for our UPROM II EPROM programmer (marketed in the U.S. by GIMIX) which will enable it to program 27512 devices from INTEL and AMD. The FLEX version will be released shortly with the OS9 versions following in about 8 weeks. This latest enhancement enables this product to handle the full spectrum of EPROMS from 1K x 8 right up to the latest 64K x 8.

Yours faithfully,

William C. Dickinson

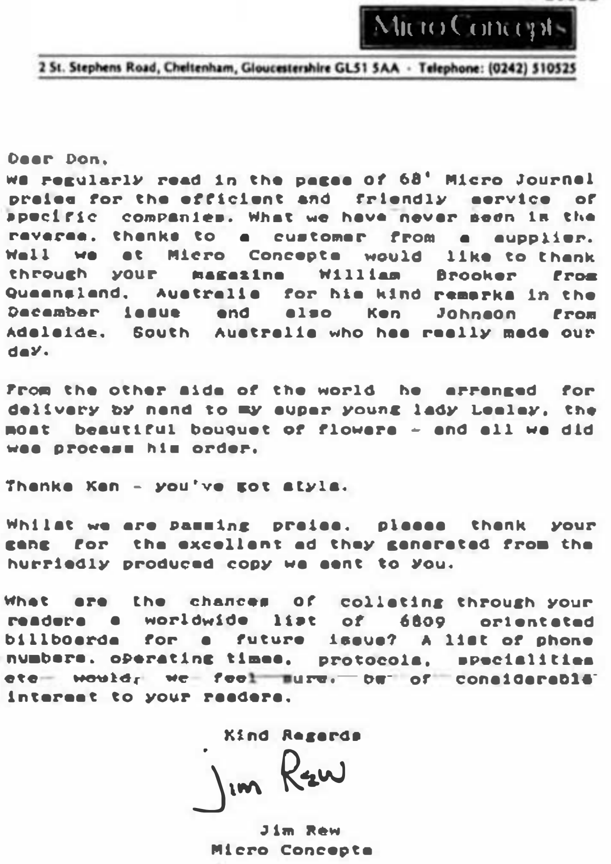
William C. Dickinson
Director



Dear CPI:

Per your letter dated June 28,1985 I am returning my copy of FMATE for updating to version 3.0. Due to a problem with the disk drive that I purchased with FMATE this disk will no longer boot up. I started up the system with my old drives and had made copies before I had the problem so I didn't need the original disk. Data-Comp repaired the disk drive however and it is still working fine. The prompt courteous service was greatly appreciated.

Best Wishes
R.E.BASSETT
4125 MARSHALL
WHEATRIDGE, COLO.
80033



Micro Concepts

2 St. Stephens Road, Cheltenham, Gloucestershire GL51 5AA · Telephone: (0242) 510525

Dear Don,

We regularly read in the pages of 68' Micro Journal praise for the efficient and friendly service of specific companies. What we have never seen is the reverse, thanks to a customer from a supplier. Well we at Micro Concepts would like to thank through your magazine William Brooker from Queensland, Australia for his kind remarks in the December issue and also Ken Johnson from Adelaide, South Australia who has really made our day.

From the other side of the world he arranged for delivery by hand to my super young lady Lesley, the most beautiful bouquet of flowers - and all we did was process his order.

Thanks Ken - you've got style.

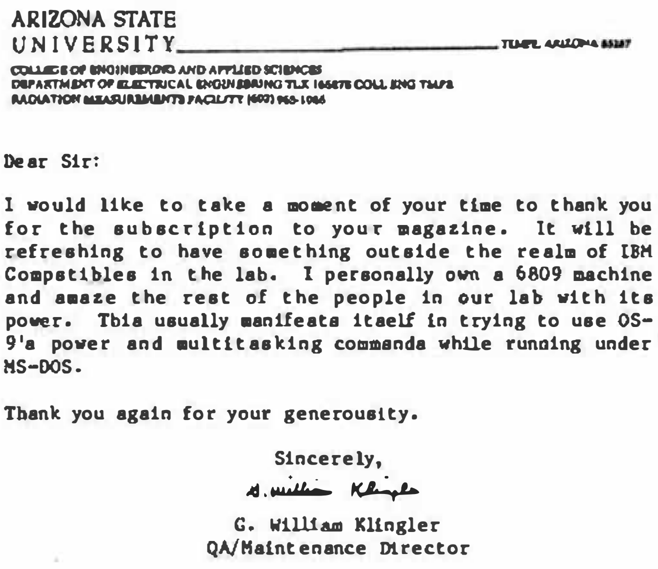
Whilst we are passing praise, please thank your gang for the excellent ad they generated from the hurriedly produced copy we sent to you.

What are the chances of collating through your readers a worldwide list of 6809 orientated billboards for a future issue? A list of phone numbers, operating times, protocols, specialities etc would, we feel sure, be of considerable interest to your readers.

Kind Regards

Jim Rew

Jim Rew
Micro Concepts



ARIZONA STATE
UNIVERSITY TEMPE, ARIZONA 85287

COLLEGE OF ENGINEERING AND APPLIED SCIENCES
DEPARTMENT OF ELECTRICAL ENGINEERING TLX 165676 COLL ENG TMPE
RADIATION MEASUREMENTS FACILITY (602) 965-1066

Dear Sir:

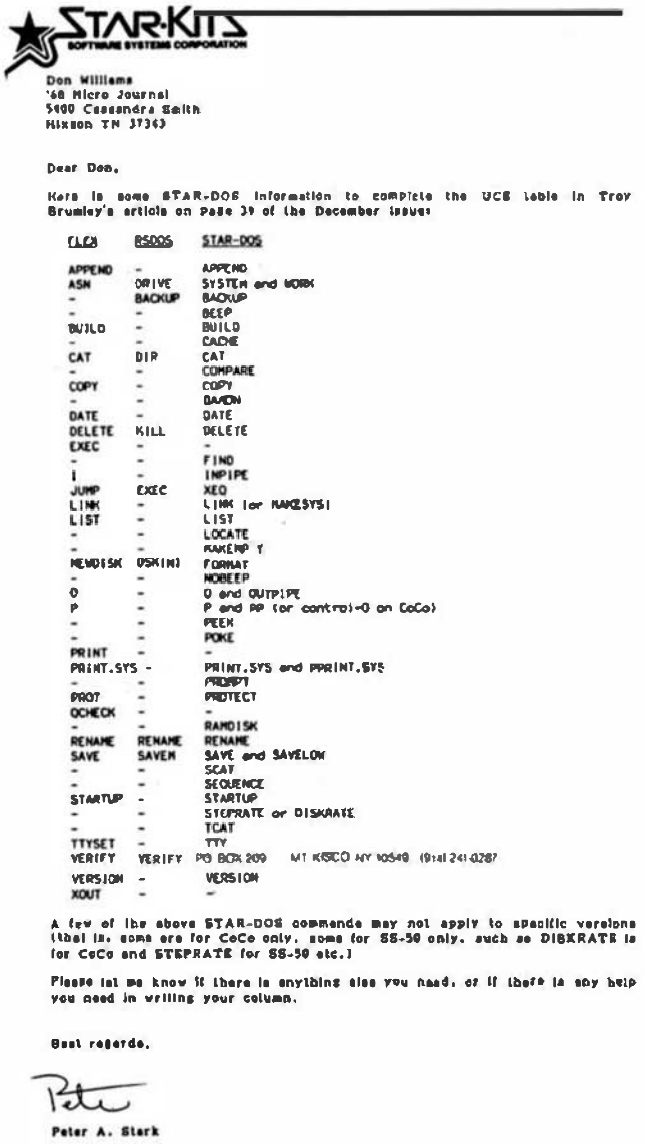
I would like to take a moment of your time to thank you for the subscription to your magazine. It will be refreshing to have something outside the realm of IBM Compatibles in the lab. I personally own a 6809 machine and amaze the rest of the people in our lab with its power. This usually manifests itself in trying to use OS-9's power and multitasking commands while running under MS-DOS.

Thank you again for your generousity.

Sincerely,

[illegible]

G. William Klingler
QA/Maintenance Director



STAR-KITS
SOFTWARE SYSTEMS CORPORATION

Don Williams
'68 Micro Journal
5900 Cassandra Smith
Hixson TN 37343

Dear Don,

Here is some STAR-DOS information to complete the UCS table in Troy Brumley's article on page 39 of the December issue:

FLEX	RSDOS	STAR-DOS
APPEND	-	APPEND
ASN	DRIVE	SYSTEM and WORK
-	BACKUP	BACKUP
-	-	BEEP
BUILD	-	BUILD
-	-	CACHE
CAT	DIR	CAT
-	-	COMPARE
COPY	-	COPY
-	-	DAMON
DATE	-	DATE
DELETE	KILL	DELETE
EXEC	-	-
-	-	FIND
I	-	INPIPE
JUMP	EXEC	XEQ
LINK	-	LINK (or MAKESYS)
LIST	-	LIST
-	-	LOCATE
-	-	MAKEMP Y
NEWDISK	DSKINI	FORMAT
-	-	NOBEEP
O	-	O and OUTPIPE
P	-	P and PP (or control-O on CoCo)
-	-	PEEK
-	-	POKE
PRINT	-	-
PRINT.SYS	-	PRINT.SYS and PPRINT.SYS
-	-	PROMPT
PROT	-	PROTECT
QCHECK	-	-
-	-	RAMDISK
RENAME	RENAME	RENAME
SAVE	SAVEM	SAVE and SAVELOW
-	-	SCAT
-	-	SEQUENCE
STARTUP	-	STARTUP
-	-	STEPRATE or DISKRATE
-	-	TCAT
TTYSET	-	TTY
VERIFY	VERIFY	PO BOX 209 MT KISCO NY 10549 (914) 241-0287
VERSION	-	VERSION
XOUT	-	-

A few of the above STAR-DOS commands may not apply to specific versions (that is, some are for CoCo only, some for SS-50 only, such as DISKRATE is for CoCo and STEPRATE for SS-50 etc.)

Please let me know if there is anything else you need, or if there is any help you need in writing your column.

Best regards,

[illegible]

Peter A. Stark

Computer Publishing Center
68' Micro Journal
5900 Cassandra Smith Road
Hixson, Tennessee 37343

149 Lewis Road
Belmont, Mass. 02178
(617) 484-5643

Dear Editor:

I had two problems with Ron Voigts' directory alphabetizer program which appears in the November "Basic OS-9" column, although the program appears to be flawless.

The first problem relates to the type font used in 68' Micro Journal. There isn't enough (if any) difference between lower-case l ("el") and the number 1 (one). Only after looking up each of the system calls in the C manual, did I discover that the second argument to "lseek" is "0L" rather than "01" as I originally transcribed it. Use of a better font or use of upper-case L would have made this clearer, although I would not have learned as much.

The second problem I encountered is due to a bug in the C run-time routine "_stkcheck". After correcting the typographic problem, the program correctly alphabetized a short directory, but crashed OS-9 at exit. When run on a large directory, the program hung up completely before rewriting the alphabetized directory.

This problem caused me to look into the C memory management scheme. The statement in the Tandy C manual that "the average programmer need not be concerned with details of memory management" is patently false. Space for "auto" variables is allocated from the stack at run time. If the space required is larger than the default 1K-byte space (such as is the case in the directory alphabetizer), "_stkcheck" should detect a stack overflow condition and terminate execution with the stack overflow message.

If, however, the data area is allocated in low memory and the absolute value of the requested size is larger than contents of the S register, the stack check routine, as coded by Microware, thinks that there is enough space. In fact, in this case, "_stkcheck" thinks that almost the entire 64K memory space is available for stack use, thus causing (at least) system data in low memory to be overlaid with program data.

I have coded a fix for the problem which I believe works in all cases, but I must point out that I do not have much assembly language experience on the 6809. "_stkcheck" is contained in the "SOURCES/SYS/cstart.a" module on the C library disk. Place the following fix between the "_stkcheck" entry label and the first instruction of the routine "leax d,s":

```
        pshs d     save contents of d
        tfr s,d    s -> d
        addd ,s    s + d -> d
        bcc fsterr  branch if abs(d) > s
        puls d     restore contents of d
```

You can save one instruction by sandwiching the fix around the "leax d,s" instruction as follows:

```
        pshs d     save contents of d
        leax d,s   originally first instr of _stkchek
        tfr s,d    s -> d
        addd ,s++ d + s -> d
        bcc fsterr  branch if abs(d) > s
```

In this case, the contents of D passed to "_stkcheck" are not preserved.

It seems that any program that has more than 1000 bytes of "auto" variables would encounter this problem (at least on the Color Computer). I am surprised that I have not seen any mention of the problem in any of the magazines (68' Micro Journal and The Rainbow).

Sincerely,

David A. Lynde

David A. Lynde

MICRONICS
RESEARCH CORP. 33383 LYNN AVENUE, ABBOTSFORD, BRITISH COLUMBIA, CANADA, V2S 1E2
(604) 859-7005
Microcomputers - Hardware and Software
GIMIX® Sales, Service and Support

Dear Don,

Last time around, I gave an introduction to the inner workings of XBASIC, and promised a further discussion on "tokens". I'll continue using the old FLEX2 version as a model, as this is the one on which I did all the detailed analysis, and I still have all my notes to hand.

OK then, I guess I can assume that most readers know that XBASIC consists of "commands" and "statements". Commands are those instructions which will only execute from the READY prompt - such as RUN, LIST, SAVE, LOAD, etc - but will not execute if they appear in a program-line. Statements, on the other hand, will execute either directly, or as part of a program-line. Whether "command" or "statement", one of the first things that XBASIC does when it encounters an instruction in a program-line is to convert it to a code-number - called a "token" - as we saw in last month's example, where "=" had got converted to its token, 69. For instance, the token for GOTO is 01, GOSUB is 02, RESUME is 03, and so on.

Why "tokenise"? The main reason is speed of execution. Without tokens, let's suppose XBASIC meets the statement PRINT in a program-line. Normally it would scan through a Statement-Table until it identified the word PRINT, which would then point to the address of the PRINT subroutine to be executed. With tokens, however, all occurrences of the word PRINT have been replaced with its token 07, so now when XBASIC comes across an 07, it "knows" that the 7th location in a Statement Jump-Table is the address of the PRINT routine, and BINGO, it can go straight to work without having to do an exhaustive table-search each time.

Now we come to one of my major "beefs" about XBASIC. For some reason - maybe to confuse the enemy - TSC has chosen to encode all commands and statements before tokenising them. That is to say, it will, for example, first "hash" the ASCII codes for SAVE (53,41,56,45) into the codes 8C,68,92,70, then search through a hashed command table in order to identify this new code, and finally tokenise it from the table. For those who are interested, the hashing procedure involves doubling the ASCII code and subtracting 1A from the result. Thus E (ASCII 45) becomes 8A by doubling, and 70 by subtracting 1A.

I think the reason TSC does this is to make it harder for a hacker to identify where these tables lie, as a HEX-ASCII dump would show none of the familiar words in the ASCII section. In that case, why not do the same with EDIT or ASMB? Maybe TSC would care to enlighten us? In fact, it deters a dedicated hacker just a teensy bit, but as far as I can see it has no other virtue at all. In effect, all this hashing and dehashing can only serve to slow down XBASIC's operation a little.

Obviously then, if we went through these tables and re-wrote the commands and statements in normal ASCII we could save XBASIC a lot of time in encoding and decoding the hashed forms. It goes without saying that in addition to converting the tables, we would also have to NOP out the hashing and dehashing routines.

Let's try that. Now that we know how to hash a word like SAVE, let's scan memory for the hashed sequence 8C,68,92,70. In my old FLEX2 version this occurs at location 0682. So we proceed through the table, "dehashing" as we go, until we come to command SCALE, with token 0E, which ends the table. Now for our first surprise. What's this weird command with token 0D???? You didn't know that XBASIC has a command T[^]@, did you? It's certainly not mentioned in the Manual. If you're curious as to what it does, try entering it in response to the READY prompt. I won't spoil your fun by giving away the secret right here.

Now we have to dehash the Statement-Table in similar manner. The first statement (token 01) is GOTO, which you should readily hash into the sequence 74,84,8E,84 and locate it in memory (in my example this occurs at address 0FF3). Here you will dehash all the way through the Statement-Table, which runs straight into the Function-Table (commencing with FN - token 34) and into the Operator-Table, which hashes the 3 logical operators NOT, AND and OR, ending at token 60. The rest of this table is taken up with other symbols, such as >, <, =, (, ?, etc., which TSC didn't bother to hash. Note, by the way, that the token for "?" is the same as that for PRINT.

Finally, we must locate the hashing and de-hashing sections, and get rid of them. The hashing section can be identified by the code

```
   48      ASL A      Multiply by 2
   80 1A   SUB A #$1A
```

and the de-hashing section by its inverse

```
   8B 1A   ADD A #$1A
   44      LSR A      Divide by 2
```

Replacing these 6 bytes with NOPs completes the whole operation.

EXCEPT FOR THIS SUGGESTION. I like to standardise my commands. By that I mean that if I want to exit to my Monitor program, I don't want to be left wondering whether I should enter MON, or is it EXIT, or maybe QUIT, or perhaps even PATCH (which was the correct entry for Uiterwyk's 8K BASIC). So ... while I was about this remodelling business, I decided to standardise on the command MON to return to Monitor. Makes sense, and I would suggest that program writers develop a set of conventional commands such as MON for exiting to MONITOR, FLEX for FLEX, and so on. Of course, by changing XBASIC's "EXIT" to "MON" we'll have to bubble-down the saved byte to the end of the table, where we'll leave it as an extra 00. Don't forget to amend all references to EXIT to read MON in your XBASIC manual, if you also decide to go this route.

That's it for now. See you next time.

MICRONICS
RESEARCH CORP.

Microcomputers - Hardware and Software
GIMIX® Sales, Service and Support

33383 LYNN AVENUE,
ABBOTSFORD,
BRITISH COLUMBIA,
CANADA, V2S 1E2

Dear Don,

Well, I guess I've just got to keep rambling on a little while longer on the subject of XBASIC, as I've received 2 YESes so far, and (guess what?) not a single NO ... so the majority wins!!!

LESSON 3.

To clear up a point (I guess I took it for granted last time) I should first mention that when I say to NOP out a bit of code, the appropriate HEX bytes should be replaced with 01 in a 6800 system, and with 12 in a 6809. I'd also like to clear up one further point - XBASIC will not actually **execute** a program any faster as a result of converting those tables to ASCII, as all statement-lines have been tokenised by RUN time. Only operations involving converting lines to a tokenised form (or conversely, de-tokenising them back to their original form) will be speeded up. This includes operations such as LIST, LOAD or SAVE.

OK ... let's move up to the 21st century with 6809 XBASIC, and repeat a mod which appeared in a long-ago issue of 68MJ. I think it bears repeating here, if only for the benefit of newcomers to 68xx computing. This mod gets rid of those tiresome "ERROR #73 AT LINE xxxx" type of error-messages, and displays the much more readable "ILLEGAL EXPRESSION AT LINE xxxx". In order for this to work, though, you will need the expanded ERRORS.SYS on Service-Disk #19, or key it all in yourself. Somewhere in the vicinity of XBASIC's address $0B77, you should be able to locate the following sequence, where xx is some code :

XBASIC	BASIC	REPLACE WITH
30 8D 00 xx	30 8D 00 xx	B7 CC 20
17 F9 xx	17 F9 xx	BE CC 1F
30 C8 xx	30 C8 xx	BD CD 3F
17 xx xx	5F	BD CD 24
	17 xx xx	12 (12)

The extra 12 (NOP) will be needed only for BASIC, not for XBASIC. I implemented this on my system long after I moved up to 6809, so I'm not sure what the equivalent code would look like in 6800 systems ... sorry!

Some further thoughts. XBASIC's Instruction-Manual contains two major errors - in its explanations of the STOP statement and the RND function.

When a program encounters the STOP statement, and is then restarted with the CONT command, **it resumes at the LINE following the STOP** and not at the STATEMENT following the STOP, thus ignoring any statements on the same line following the STOP. Depending on the nature of the program-bug you are trying to pin down, it is sometimes advantageous to put the STOP somewhere **inside** a program-line, and at other times at the end of a line.

RND(X) actually generates a number in the range 0 through 0.9999999......, **not 0 through 1**, and so the formula given (to generate random **integers** over a desired range) should be amended to read :

```
     X% = (ML - MS + 1) * RND(0) + MS
```

otherwise it will not generate ML. But leave it as is if you wish to generate fractional numbers in the range MS up to, **but not including**, ML.

While on the subject of random numbers .... Don't say I didn't warn you. Right at the beginning of this letter I told you I was going to ramble on. Anyway, I've always found it a good idea to incorporate a known random-seed in the RND-generator, by inserting something like the following somewhere near the start of any XBASIC program (particularly a Game) which uses RND(X) :

```
50 INPUT "Please enter a random number between 0 and
   99999 (0 to allow the computer to choose) ",X
60 X = RND(-X)
```

This is particularly useful if the program has a bug, as it is possible to repeat the game sequence, commencing with the original random seed (unless of course a response of 0 was made). It can also be used in the event of a really hard-fought game, where you lost to the computer by a VERY small margin, and would like to try your hand all over again using slightly different tactics. Decimal fraction numbers are equally acceptable, eg 1234.56789, as a response.

Now my thoughts are really running free! I guess it was the word "Game" just above. As you well know, lots of games include rules or instructions for play, and usually ask the player whether he needs these instructions. They take up a lot of room in memory, and if they occur in the wrong place in a program they help to slow down execution. Besides which, after a while you don't need them any more, and yet you don't want to throw them out in case a new player comes along who's not familiar with the whole shebang.

I am now beginning to remove all instructions from programs, by judicious use of the SPLIT utility. After splitting them off, I use the EDITOR to remove all items relating to XBASIC, such as line-numbers, PRINT statements, etc., leaving only the text which was originally enclosed in quotes, (deleting the quotes themselves) and rename the file to <game.DOC> or <game.INF>. Then I go back to the game itself, and set up a sequence something along the lines of :

```
50 INPUT "Do you want instructions (Y or N) ",Q$
60 IF Q$ = "N" OR Q$="n" GOTO 80
70 EXEC,"TTYSET PS=Y: LIST game.DOC: TTYSET PS=N"
80 rest of game-program ......
```

Talking about INPUT ... Each of us knows that PRINT statements can be chained, eg :

```
50 PRINT "Your name is ";N$;" and your age is";A%; etc
```

but how many of us know that INPUT statements can also be chained? Thus :

```
50 INPUT "What is your name ",N$;"Your age ",A%;  etc
```

I had never seen this mentioned anywhere, (other than in a DEC BASIC-PLUS Manual) and decided to try it out during one of my exploration forays into XBASIC. Lo and behold, it worked!!!

If you really want to speed up your XBASIC programs, you should make more use of XBASIC's COMPILE command, by first saving your program in the regular manner and then executing COMPILE "file-name". The reason you need the SAVEd version as well is that you can't edit a COMPILEd program. It will have the extension .BAC, and can be fired up from FLEX with the command-line 'XBASIC file-name' or from XBASIC itself with the command 'RUN file-name'. Now where was I? Ah, yes

... a .BAC file normally takes up a lot less room on disk or in memory, and also loads considerably faster than the regular .BAS versions.

But have you ever tried to get XBASIC to LIST a .BAC file which it has in memory? It will snap right back with "SOURCE NOT PRESENT", and yet we know it's really there, because it has no trouble executing it. Would you like to see it LISTed? Someone out there said YES (I hope) so here goes ...

First load XBASIC, then execute MON (or EXIT if you didn't carry out my last month's mods). Now, somewhere in the vicinity of address $0900 ($0908 for XBASIC version 19, and $0905 for version 24) you should look for code something like 6D C8 57 27 xx 86 40. The important byte is the 6D - - change this to 39, then execute a jump to XBASIC's Warm-Start with GMXBUG's J 3, or its equivalent in your Monitor.

Now LOAD your .BAC program with LOAD "file-name.BAC", and LIST. Did you ever see such a compact program format? No wonder it takes up so much less room in memory (or on disk). Now you see why you can't change it if it has a bug in it. You must identify the buggy line, debug the SAVEd .BAS version, and re-COMPILE a new .BAC file.

Removing instructions, or COMPILing, will most often allow you to run those REALLY large programs which won't quite fit into memory otherwise, and is preferable to removing those essential REMs in your program.

What did you think of the command T[^]@ in XBASIC? Pretty useless for our needs, don't you agree? But quite handy if you want to replace it with another command, such as EDIT, which is what got me started on this XBASIC thing in the first place.

Hmm ... seems to me I should cut off about here, before I take up the whole issue with a single letter. In the meantime, I'd appreciate some feedback from you folks out there. For instance, would you prefer a structured, tutorial type of letter from me, or a rambling, informal smorgasbord like the above?

Don Williams,
68 Micro Journal,
5900 Cassandra Smith Road,
Hixson, TN 37343

Sincerely,

[illegible]

R. Jones
President

Don,

Where are the MC10 users? Has everyone given up on their POCO COCO? There is still much to be said for this Radio Shack reject. How about 44K of memory for starters. If you have a MC10 and want to use it for something other than a paper weight, drop me a line at the address below. Let's get a MC10 users group started!!

H. Francisco
10226 N. 29th St.
Tampa, FL 33612

Editor's Note: O.K. here is my "dropped line". I've got a couple, but not much to do with them. Really not even heavy enough to make paper weight (if wind is blowing). Just wait and there will soon be some bigger (and heavier) Radio Shack rejects. They will make much better paper weights.
  If you come up with something useful, I really would like to know. It seems to be such a neat package. But there is just not much support software for it. Guess there never will be any offered by RS. Please keep me posted on this.

DMW

- - -

LLOYD I/O
TM INC.

Lloyd I/O is a computer engineering corporation providing software and hardware products and consulting services.

19535 NE GLISAN • PORTLAND, OR 97230 (USA)
PHONE: (503) 666-1097 • TELEX: 910 380 5448 LLOYD I O

New Product!

CRASMB™ CROSS ASSEMBLER NOW AVAILABLE FOR OS9/68000

LLOYD I/O announces the release of the CRASMB 8 Bit Macro Cross Assembler for Microware's OS9 disk operating system for the 68000 family of microprocessors. In recent increasing demand for the OS9/68000 version of CRASMB, LLOYD I/O has translated its four year old CRASMB for the OS9/6809 and FLEX/6809 to the OS9/68000 environment.

CRASMB supports assembly language software development for these microprocessors: 1802, 6502, 6800, 6801, 6303, 6804, 6805, 6809, 6811, TMS 7000, 8048/family, 8051/family, 8080/85, Z8, and the Z80. CRASMB is a full featured assembler with macro and conditional assembly facilities. It generates object code using 4 different formats: none, FLEX, Motorola S1-S9, and Intel Hex. Another format is available which outputs the source code after macro expansion, etc. CRASMB allows label (symbols) length to 30 characters and has label cross referencing options.

CRASMB for OS9/68000 is available for $432 in US funds only. It may be purchased with VISA/MASTERCHARGE cards, checks, US money orders, or US government (federal, state, etc.) purchase orders. NOTE: please add $5 shipping in the USA and use your street address for UPS shipments. Add $30 for all overseas orders. CRASMB for OS9/6809 and FLEX/6809 cost $399 plus shipping.

You may contact Frank Hoffman at LLOYD I/O, 19535 NE Glisan, Portland, Oregon, 97230. Phone: (503) 666-1097. Telex: 910 380 5448, answer back: LLOYD I O. Easylink: 62846110. See list of distributors below.

VISA, MC, COD, CHECKS, ACCEPTED

USA:	LLOYD I/O (503 666 1097),	S.E. MEDIA (800 338 6800)
England:	Vivaway (0582 423425),	Windrush (0692 405189)
Germany:	Zacher Computer (65 25 299),	Kell Software (06203 6741)
Australia:	Paris Radio Electronics (344 9111)	
Japan:	Microboards (0474) 22-1741	Seikou (03) 832-6000
Switzerland:	Elsoft AG (056 86 27 24)	
Sweden:	Micromaster Scandinavian AG (018 - 138595)	

K-BASIC, DO, SEARCH and RESCUE UTILITIES, PATCH, CRASMB, and CRASMB 16.32 are trademarks of LLOYD I/O. OS9 is a ™ of Microware. FLEX is a ™ of TSC

Classified Advertising

Tano Outpost II, 56K, 2 5"DSDD Drives, FLEX, MUMPS $895.
MICROKEY 4500 Single Board Computer, Target 128K RAM, FLEX, FORTH, with optional 6502 CPU & ROMS as advertised on p.51 Dec.84 68 Micro Journal. $2300.
LSI 68008 CPU Card with Digital Research CPM/68K $350.
1-PT-69 complete with Dual 5" DSDD Disk System and Controller, includes FLEX DOS.
TELETYPE Model 43 PRINTER - with serial (RS232) interface, and full ASCII keyboard. LIKE NEW - New cost $1295.00 - ONLY $559.00 ready to run.
S/09 with Motorola 128K RAM, 1-MPS2, 1-Parallel Port, MP-09 CPU Card $1990. 1-DMAF2 Dual 8" Drives with Controller $2190. 1-CDS1 20 Meg Hard Disk System with Controller $2400.

I will accept any reasonable counter-offer !!

Call Tom (615) 842-4600 M-F 9 A.M. to 5 P.M. E.S.T.

***

SWTPC 6809 S-09 Computer System with 40 MEG Hard Disk, Dual 8" Drivea, 6 Serial Cards, 256 K Memory. Runs UniFLEX. In Good Running Order. $2000.00 or BEST 1-219-259-8511 EXT 362 DAYS 1-219-674-9654 EVENINGS.

***

Several 2MHZ 64K Static Ram cards, $75 each. Misc. SS50 and SS30 cards. Call with your needa. (703)273-6629 evenings.

***

WANTED!! GIMIX #68 - Controller, will trade GIMIX 6809+ CPU and SSB DCB-4A Controller.
Dick Ollendorf (201)852-6764 AFTER 7PM

***

WANTED: Two Data Systems 68 motherboards. Prefer un-assembled. Will pay reasonable price. Call Mickey (after 7pm central) (615)-797-9963.

SOFTWARE DEVELOPERS!

YOU'VE JUST BEEN GIVEN THE BEST REASON YET TO GET OUR 68000/UNIX® DEVELOPMENT SYSTEM

THE VAR/68K® SERIES

VK-5XW20 $10,100
Includes: Terminal, 20 Mb hard disk, 512K RAM, 8 ports and REGULUS®

VK-5XW20T20 $12,900
Includes: all of above, plus 20 Mb tape streamer

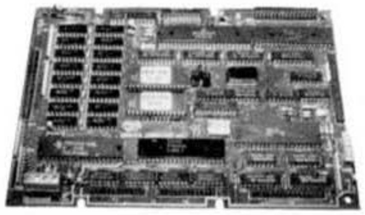
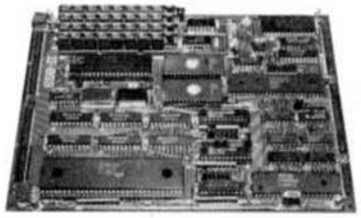
RESELLERS!

IBM PC/OS9 Compatibility

Smoke Signal can add IBM PC capability to your OS9 system for as little as $1195 plus software.

VAR/68K is a registered trademark of Smoke Signal
REGULUS is a registered trademark of Alcyon Corp
UNIX is a registered trademark of Bell Laboratories

TO OBTAIN YOUR VAR/68K AT THESE LOW PRICES, CONTACT:
SMOKE SIGNAL
31336 VIA COLINAS
WESTLAKE VILLAGE, CA 91362
(818) 889-9340 / Telex 910 494 4965

Both boards designed to the 5¼ inch disk drive footprint

ESB-I **$495**

- 68008 microprocessor - 8Mhz
- 128K DRAM
- Up to 64K ROM
- Two asynchronous serial ports
- Two 8 bit parallel ports
- Floppy disk controller / up to four 5¼" drives

ESB-II **$595***

- 68000/68010 8, 10, or 12 Mhz
- Up to 1Mb DRAM
- Up to 128K ROM
- Two asynchronous serial ports
- 24 bit parallel port
- One 24 bit timer / one 16 bit timer
- Floppy disk controller / up to four 5¼" drives
- OS9/68K Operating System

*68000 - 8 Mhz w/128K DRAM version
Consult factory for expanded configurations.

***COMING SOON:* ESB-IR**

68008 - 10 Mhz * 512K DRAM * 3½" diskette footprint * runs OS9/68K

EMERALD COMPUTERS INC.

(503) 620-6094 - Telex: 311593
- 16515 S.W. 72nd Avenue - Portland, Oregon 97224

COMPILER EVALUATION SERVICES
By: Ron Anderson

The S.E. MEDIA Division of Computer Publishibg Inc., is offering the following SUBSCRIBER SERVICE:

COMPILER COMPARISON AND EVALUATION REPORT

Due to the constant and rapid updating and enhancement of numerous compilers, and the different utility, appeal, speed, level of communication, memory usage, etc., of different compilers, the following services are now being offered with periodic updates.

This service, with updates, will allow you who are wary or confused by the various claims of compiler vendors, an opportunity to review comparisons, comments, benchmarks, etc., concerning the many different compilers on the market, for the 6809 microcomputer. Thus the savings could far offset the small cost of this service.

Many have purchased compilers and then discovered that the particular compiler purchased either is not the most efficient for their purposes or does not contain features necessary for their application. Thus the added expense of purchasing additional compiler(s) or not being able to fully utilize the advantages of high level language compilers becomes too expensive.

The following COMPILERS are reviewed initially, more will be reviewed, compared and benchmarked as they become available to the author:

PASCAL "C" GSPL WHIMSICAL PL/9

Initial Subscription - $ 39.95
(includes 1 year updates)
Updates for 1 year - $ 14.50

S.E. MEDIA - C.P.I.
5900 Cassandra Smith Rd.
Hixson, Tn. 37343
(615) 842-4601

OS-9™ SOFTWARE

SDISK—Standard disk driver module allows the use of 35, 40, or 80 track double sided drives with COCO OS-9 plus you can read/write/format the OS-9 formats used by other OS-9 systems. **$29.95**

SDISK + BOOTFIX—As above plus boot directly from a double sided diskette **$35.95**

FILTER KIT #1—Eleven OS-9 utilities for "wild card" directory lists, copies, moves, deletes, sorts, etc. Now includes disk sector edit utility also. **$29.95** ($31.95)

FILTER KIT #2—Macgen command macro generator builds new commands by combining old ones with parameter substitution, 10 other utilities. **$29.95** ($31.95)

HACKER'S KIT #1—Disassembler and related utilities allow disassembly from memory, file. **$24.95** ($26.95)

PC-XFER UTILITIES —Utilities to read/write and format MS-DOS™ diskettes on CoCo under OS-9. Also transfer files between RS disk basic and OS-9 (requires sdisk). **$45.00**

BOLD prices are CoCo OS-9 format disk, other formats (in parenthesis) specify format and OS-9 level. All orders prepaid or COD, VISA and MasterCard accepted. Add $1.50 S&H on prepaid, COD actual charges added.

SS-50C
1 MEGABYTE RAM BOARD

Full megabyte of ram with disable options to suit any SS-50 6809 system. High reliability, can replace static ram for a fraction of the cost, $895 for 2 Mhz or $995 for 2.25 Mhz board assembled, tested and fully populated. (Add $6 shipping and insurance, quantity discounts available.)

D.P. Johnson, 7655 S.W. Cedarcrest St.
Portland, OR 97223 (503) 244-8152
(For best service call between 9-11 AM Pacific Time.)

OS-9 is a trademark of Microware and Motorola Inc.
MS-DOS is a trademark of Microsoft, Inc.

68000 68020 68010

68008 6809 6800

Write or phone for catalog.

AAA Chicago Computer Center
120 Chestnut Lane — Wheeling IL 60090
(312) 459-0450

Technical Consultation available most weekdays from 4 PM to 6 PM CST

Introducing........

S-50 BUS/68XX

Board and/or Computer

Terminals-CRTs-Printers

Disk Drives-etc.

REPAIRS

This

NOW AVAILABLE TO ALL S50/68XX USERS

The Data-Comp Division of CPI is proud to announce the availability of their service department facilities to 'ALL' S50 Bus and 68XX users. Including all brands, SWTPC - GIMIX - SSB - HELIX and others, including the single board computers. * Please note that kit-built components are a special case, and will be handled on an individual basis, if accepted.

1. If you require service, the first thing you need to do is call the number below and describe your problem and **confirm a Data-Comp service & shipping number!** This is very important, Data-Comp will not accept or repair items not displaying this number! Also we cannot advise or help you troubleshoot on the telephone, we can give you a shipping number, but NO advice! Sorry!

2. All service shipments must include both a minimum $40.00 estimate/repair charge and pre-paid return shipping charges (should be same amount you pay to ship to Data-Comp).

3. If you desire a telephone estimate after your repair item is received, include an additional $5.00 to cover long distance charges. Otherwise an estimate will be mailed to you, if you requested an estimate. Estimates must be requested. Mailed estimates slow down the process considerably. However, if repairs are not desired, after the estimate is given, the $40.00 shall constitute the estimate charge, and the item(s) will be returned unrepaired providing sufficient return shipping charges were included with item to be serviced. Please note that estimates are given in dollar amounts only.

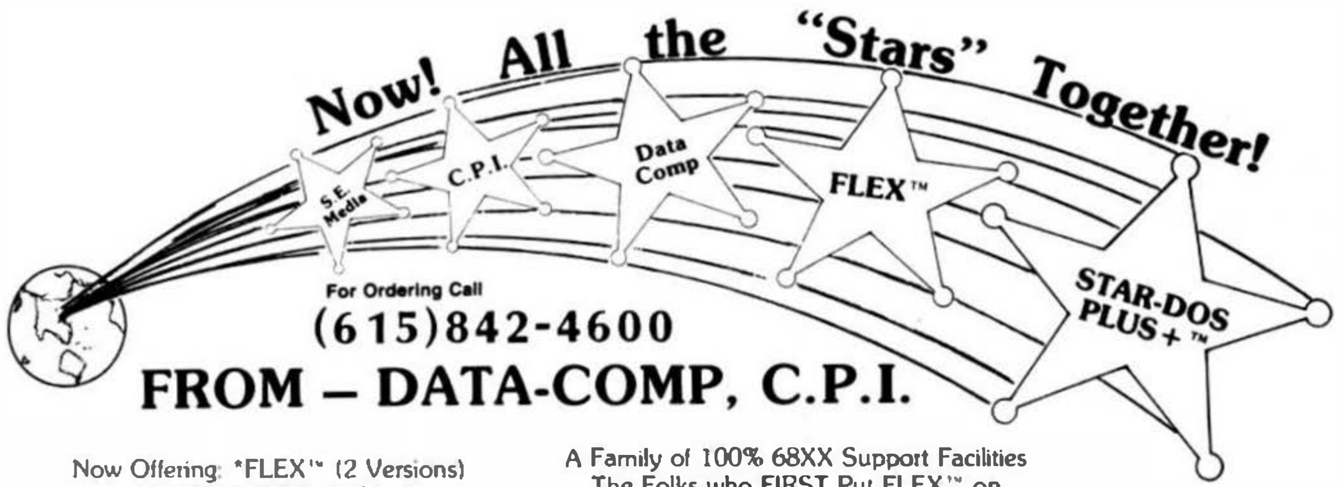
4. Data-Comp service is the oldest and most experienced general S50/68XX service department in the world. We have over $100,000.00 in parts in stock. We have the most complete set of service documents for the various S50/68XX systems of anyone - **YET, WE DO NOT HAVE EVERYTHING!** But we sure have more than anyone else. We repair about 90% of all items we receive. Call for additional information or shipping instructions.

Not This

DATA-COMP
5900 Cassandra Smith Rd.
Hixson, TN 37343

☎ **(615)842-4607**

Telex 5106006630



For Ordering Call

(615)842-4600

FROM – DATA-COMP, C.P.I.

Now Offering: *FLEX™ (2 Versions)
AND *STAR-DOS PLUS+™

A Family of 100% 68XX Support Facilities
The Folks who FIRST Put FLEX™ on
The CoCo

FLEX-CoCo Sr.
with TSC Editor
TSC Assembler
Complete with Manuals
Reg. $250.00 **Only $79.95**

STAR-DOS PLUS+
- Functions Same as FLEX
- Reads - writes FLEX Disks
- Run FLEX Programs
- Just type: Run "STAR-DOS"
- Over 300 utilities & programs to choose from.

$34.50

FLEX-CoCo Jr.
without TSC
Editor & Assembler
$49.95

PLUS

ALL VERSIONS OF FLEX & STAR-DOS• INCLUDE

TSC Editor
Reg $50.00
NOW $35.00

+ Read-Write-Dir RS Disk
+ Run RS Basic from Both
+ More Free Utilities
+ External Terminal Program
+ Test Disk Program
+ Disk Examine & Repair Program
+ Memory Examine Program
+ Many Many More!!!

TSC Assembler
Reg $50.00
NOW $35.00

CoCo Disk Drive Systems

2 THINLINE DOUBLE SIDED DOUBLE DENSITY DISK DRIVES SYSTEM WITH POWER SUPPLY,CABINET,DISK DRIVE CABLE,J&M NEW DISK CONTROLLER JFD-CP WITH J-DOS,RS-DOS OPERATING SYSTEMS.	$469.95

* Specify What CONTROLLER You Want J&M, or RADIO SHACK

THINLINE DOUBLE SIDED DOUBLE DENSITY 40 TRACKS	$129.95

Verbatim Diskettes

Single Sided Double Density	$ 24.00
Double Sided Double Density	$ 24.00

Controllers

J&M JFD-CP WITH J-DOS	$139.95
WITH J-DOS, RS-DOS	$159.95
RADIO SHACK 1.1	$134.95
RADIO SHACK Disk CONTROLLER 1.1	$134.95

Disk Drive Cables

Cable for One Drive	$ 19.95
Cable for Two Drives	$ 24.95

MISC

64K UPGRADE FOR C,D,E,F, AND COCO II	$ 29.95
RADIO SHACK BASIC 1.2	$ 24.95
RADIO SHACK DISK BASIC 1.1	$ 24.95
DISK DRIVE CABINET FOR A SINGLE DRIVE	$ 49.95
DISK DRIVE CABINET FOR TWO THINLINE DRIVES	$ 69.95

PRINTERS

EPSON LX-80	$209.95
EPSON MX-70	$125.95
EPSON MX-100	$495.95

ACCESSORIES FOR EPSON

8148 2K SERIAL BOARD	$ 89.95
8149 32K EXPAND TO 128K	$169.95
EPSON MX-RX-80 RIBBONS	$ 7.95
EPSON LX-80 RIBBONS	$ 3.95
TRACTOR UNITS FOR LX-80	$ 39.95

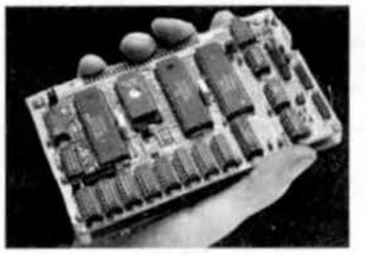
CABLES & OTHER INTERFACES
CALL FOR PRICING

DATA-COMP
5900 Cassandra Smith Rd.
Hixson, TN 37343

VISA

SHIPPING
USA ADD 2%
FOREIGN ADD 5%
MIN. $2.50

(615)842-4600
For Ordering
Telex 5106006630



512K RAM Expansion

Compact Flexible 6809 Computer

The new ST-2900 system — a complete 64K small business or hobbyist computer is only one of its many possible configurations. Among its features are:

- Small enough to hold in your hand! (Eurocard size: 3.9"x6.3")
- Three board "system" for greater versatility than single board computers.
- CPU Board — powerful 6809E processor, 16K or 64K RAM, 1K-32K EPROM, 2 RS232 serial ports with software programmable baud rates, 16 bit counter/timer. Run the CPU board all by itself, or plug your own custom board or our FDC board and/or RAM-512 board into the expansion connector.
- FDC Board — double-sided/double-density floppy disk controller with adjustment free digital data separator and write precompensation. 2 8-bit parallel ports, 2 16-bit counter/timers, prototyping area.
- **RAM-512 board** — 524,288 bytes of RAM on a 4.15"x6.3" board! Low power. Includes RAM-Disk software for FLEX or OS-9.
- FLEX and OS-9 supported — **software selectable.**
- OS-9 Conversion Package lets you use the low cost Radio Shack CoCo version of OS-9 on our ST-2900 system. Save $131 off the suggested list price of OS-9! **No programming is involved.** Supports CoCo OS-9, standard OS-9, and MIZAR OS-9/68K disk formats. Compatible with PC-XFER to let you **read/write/format MS-DOS disks!**

• CPU bare board plus EPROM	$45	OS-9 Conversion Package	$49
FDC bare board	$38	FLEX Conversion Package	$29
RAM-512 board A&T	$485	CPU + FDC + OS-9 Conversion	$119
CPU + FDC board set assembled and tested			$329

- Add $5 shipping/handling ($10 overseas). These prices are in U.S. funds. Canadian orders: call or write for prices. Terms: check, money order, VISA.

(trademarks: FLEX — Technical Systems Consultants, OS-9 — Microware & Motorola, MS-DOS — Microsoft)

RAM-512 deliveries scheduled to begin end of Sept. '85

SARDIS TECHNOLOGIES

Call or write for free brochure and complete price list (604) 255-4485

2261 E. 11th Ave. Vancouver, B.C., Canada V5N 1Z7

ARCADE 50

POWERFUL COLOR GRAPHICS
Uses the new TMS9918A Video Display procesor. High resolution 256 x 192 pixel display with 15 colors. 16K Bytes of onboard RAM does not reduce user memory. 32 graphic images can be individually moved with simple X-Y commands for smooth animation. External Video input allows subtitling. NTSC composite video output.

SOUND EFFECTS AND MUSIC
- Three AY3-8910 Programmable Sound Generators
- Nine simultaneous voices
- Three independent noise sources
- Onboard stereo amplifier drives two 8 ohm speakers

ADDITIONAL I/O CAPABILITIES
- Eight analog inputs with 8 bit resolution
- Supports four joysticks with pushbutton switches
- Eight bit parallel I/O port
- Entire unit maps into 256 bytes of memory

FBASIC

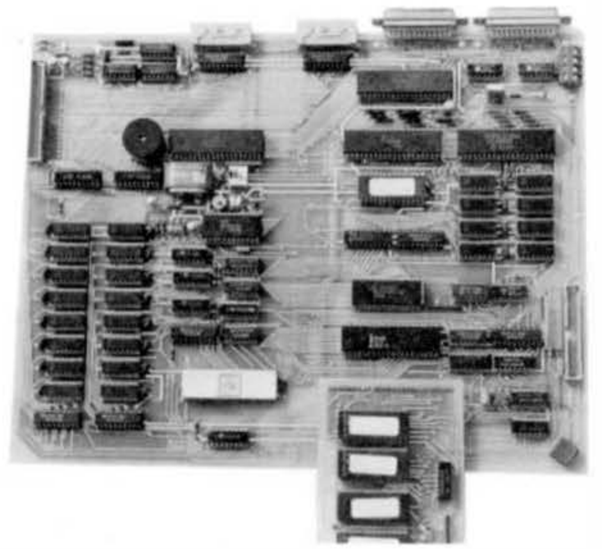
TERMINUS DESIGN INC. in conjunction with Microware Systems Corporation, is proud to announce FBASIC an enhancement of Microware's 6800/BASIC. Their last compiled BASIC has been adapted for 6809 users with added video and sound features for ARCADE 50 users. FBASIC is a true compiler that produces optimized machine language modules which are ROMable and require no Run-Time package. FBASIC requires less memory overhead and runs hundreds of times faster than BASIC interpreters. It supports standard BASIC instruction including String functions Disk I/O and fast integer arithmetic with multiple precision capability Graphics verbs and functions fully support the Arcade 50

ARCADE 50 assembled and tested	$325.00
Video and Audio connector set	15.00
4 Joystick connector set	15.00
2 Radio Shack Joysticks	24.00
Gold Molex connectors	12.00
A/BASIC for 6800	110.00
FBASIC for 6809	110.00
FBASIC (with ARCADE 50)	75.00
ARCADE 50 RGB	375.00
LA VIDEO (Motorola EXORbus)	375.00
NEW MV09 6809 Processor Board	225.00
256K Dynamic Memory Board	795.00
256K Dynamic Memory Board (w/64K)	395.00
64K Dynamic Memory Board	295.00

TERMINUS DESIGN INC
16 SCARBROUGH ROAD
ELLENWOOD, GA 30049
(404) 474-4866

[illegible]

MICROBOX II Single Board Computer



Microbox II SBC bare board (12x9.5 inch) plus eprom disc bare board (4x) inch) plus firmware and system utility disc (5.25 40 track) Supplied with full construction and operating notes.	$190
Firmware source code on disc.	$ 29
Small "C" compiler (6809) w/7220A graphic libraries.	$ 100
Small "C" compiler (6801) object.	$ 130

Prices include airmail postage.

Terms: CWO. Payment by International Money Order or MASTER-CARD.

MICRO CONCEPTS 2 ST STEPHENS ROAD CHELTENHAM GLOS. ENGLAND GL51 5AA
Telephone: (0242) 510525

Microbox II is a powerful 6809 based single board computer packed with innovative features in an easy to build form.
Running under the Flex operating system it contains 60K of dynamic ram, 8K of eprom, high resolution text and graphic displays, up to 500 sector ramdisc, up to 512 sector eprom disc, floppy disc controller, serial and parallel I/O, real-time clock and eprom programmer.
An eprom disc that looks to Flex like a standard write protected drive can be programmed with anything that would normally be on floppy - including Flex itself. A ram disc that looks like a standard unprotected disc acts as a very fast work disc. Support for two floppy drives is also on-board.
Exceptional monochrome graphic capabilities are provided by a NEC7220A graphic display controller which gives very fast drawing speeds through hardware vector, circle, rectangle, pattern and area fill generation.
The Flex operating system can be booted from any standard system disc - configuration is carried out automatically by the supplied firmware - and all the usual software can be used.
Microbox II can be controlled from a standard serial terminal a serial / parallel keyboard and video monitor or a mixture of both.

Specification:

6809E microprocessor supporting 60K of dynamic ram and 8K firmware.
7220A graphic display controller supporting 128K of dynamic ram partitioned as monochrome video display and ramdisc.
Text display of 84x24 or 108x24 characters. Or invent your own format.
Graphic display of 768x576 pixels. Very fast hardware vectors etc.
Composite video and separate video / sync outputs.
Eprom disc using four 27128 devices. An eprom programmer is on board.
Floppy disc controller for 48 or 96 tpi single/double density drives.
Two RS232 serial ports with programmable baudrates, 50 - 19200 baud.
Centronics type parallel printer port.
Parallel keyboard port.
Battery backed real-time clock/calendar.
DIP switch selection of input source, output destination and autoboot.
Additional I/O capability via user expansion buss.
100's of Microbox II's currently in use worldwide.

The firmware includes system diagnostics, utilities, graphic primatives, terminal emulator and auto-configuration that ensures that the board will boot from any standard Flex system disc.
The software includes disc formatter, printer drivers, disc allocation, alternative terminal emulators, eprom programmer routines, real-time clock support, graphic macros and demo, character set source and system equates.

*Flex is a trademark of Technical Systems Consultants.

SOFTWARE FOR 680x SYSTEMS

SUPER SLEUTH DISASSEMBLERS
EACH $99-FLEX $101-OS/9 $100-UNIFLEX
OBJECT-ONLY versions: EAC $50-FLEX,OS/9,COCO
interactively generate source on disk with labels, include xref, binary editing
specify 6800,1,2,3,5,8,9/6502 version or Z80/8080,5 version
OS/9 version also processes FLEX format object file under OS/9
COCO DOS available in 6800,1,2,3, .8,9/6502 version (not Z80/8080,5) only

CROSS-ASSEMBLERS (TRUE ASSEMBLERS, NOT MACRO SETS)
EACH $50-FLEX,OS/9,UNIFLEX ANY 3 $100 ALL $200
specify for 180x,6502,6801,6804,6805,6809,Z8,Z80,8048,8051,8085,68000
true, modular, free-standing cross-assemblers in C, with load/unload utilities
8-bit (not 68000) sources included with all cross-assemblers (for $200)

DEBUGGING SIMULATORS FOR POPULAR MICROPROCESSORS
EACH $75-FLEX $100-OS/9 $80-UNIFLEX
OBJECT-ONLY versions: EACH $50-COCO FLEX,COCO OS/9
interactively simulate processors, include disassembly formatting, binary editing
specify for 6800/1, (14)6805, 6502, 6809 OS/9, Z80 FLEX

ASSEMBLER CODE TRANSLATORS FOR 6502, 6800/1, 6809

6502 to 6809	$75-FLEX $85-OS/9 $80-UNIFLEX
6800/1 to 6809 & 6809 to position-ind	$50-FLEX $75-OS/9 $60-UNIFLEX

FULL-SCREEN XBASIC PROGRAMS with cursor control
AVAILABLE FOR FLEX, UNIFLEX, AND MSDOS

DISPLAY GENERATOR/DOCUMENTOR	$50 w/source, $25 without
MAILING LIST SYSTEM	$100 w/source, $50 without
INVENTORY WITH MRP	$100 w/source, $50 without
TABULA RASA SPREADSHEET	$100 w/source, $50 without

DISK AND XBASIC UTILITY PROGRAM LIBRARY
$50-FLEX $30-UNIFLEX/MSDOS
edit disk sectors, sort directory, maintain master catalog, do disk sorts, resequence some or all of BASIC program, xref BASIC program, etc.
non-FLEX versions include sort and resequencer only

CMODEM TELECOMMUNICATIONS PROGRAM
$100-FLEX,OS/9,UNIFLEX
OBJECT-ONLY versions: EACH $50-FLEX,OS/9
menu-driven with terminal mode, file transfer, MODEM7, XON-XOFF, etc.
for COCO and non-COCO; drives internal COCO modem port up to 2400 Baud

HARDWARE & SERVICES

5.25" DISKETTES
EACH 10-PACK $12.50-SSSD/SSDD/DSDD $20-DSQD
American-made, guaranteed 100% quality, with Tyvek jackets, hub rings, and labels

SS-50C 256K 1.5MHz MEMORY BOARDS
EACH BLANK $80 ASSEMBLED AND TESTED $350
with instruction manual, schematics, and delay line; all parts readily available

ADDITIONAL SERVICES FOR THE COMPUTING COMMUNITY

CUSTOMIZED PROGRAMMING
we will customize any of the programs described in this advertisement or in our brochure for specialized customer use or to cover new processors; the charge for such customization depends upon the marketability of the modifications.

CONTRACT PROGRAMMING
we will create new programs or modify existing programs on a contract basis, a service we have provided for over twenty years; the computers on which we have performed contract programming include most popular models of mainframes, including IBM, Burroughs, Univac, Honeywell, most popular models of minicomputers, including DEC, IBM, DG, HP, AT&T, and most popular brands of microcomputers, including 6800/1, 6809, Z80, 6502, 68000, using most appropriate languages and operating systems, on systems ranging in size from large telecommunications to single board controllers; the charge for contract programming is usually by the hour or by the task.

CONSULTING
we offer a wide range of business and technical consulting services, including seminars, advice, training, and design, on any topic related to computers; the charge for consulting is normally based upon time, travel, and expenses.

Computer Systems Consultants, Inc.
1454 Latta Lane, Conyers, GA 30207
Telephone 404-483-1717 or 4570

We take orders at any time, but please plan discussions after 6, if possible.

Contact us about catalog, dealer, discounts, and services.
Most programs in source: give computer, OS, disk size.
25% off multiple purchases of same program on one order.
VISA and MASTER CARD accepted; US funds only, please.
Add GA sales tax (if in GA) and 5% shipping.
(UNIFLEX tm Technical Systems Consultants; OS/9 tm Microware; COCO tm Tandy)

SOFTWARE FOR THE HARDCORE

** FORTH PROGRAMMING TOOLS from the 68XX&X **
** FORTH specialists — get the best!! **

NOW AVAILABLE — A variety of rom and disk FORTH systems to run on and/or do TARGET COMPILATION for
6800, 6301/6801, 6809, 68000, 8080, Z80

Write or call for information on a special system to fit your requirement.

Standard systems available for these hardware—

EPSON HX-20 rom system and target compiler
6809 rom systems for SS-50, EXORCISER, STD, ETC.
COLOR COMPUTER
6800/6809 FLEX or EXORCISER disk systems.
68000 rom based systems
68000 CP/M-68K disk systems, MODEL II/12/16

tFORTH is a refined version of FORTH Interest Group standard FORTH, faster than FIG-FORTH. FORTH is both a compiler and an interpreter. It executes orders of magnitudes faster than interpretive BASIC. MORE IMPORTANT, CODE DEVELOPMENT AND TESTING is much, much faster than compiled languages such as PASCAL and C. If Software DEVELOPMENT COSTS are an important concern for you, you need FORTH!

firmFORTH™ is for the programmer who needs to squeeze the most into roms. It is a professional programmer's tool for compact rommable code for controller applications.

™ tFORTH and firmFORTH are trademarks of Talbot Microsystems
™ FLEX is a trademark of Technical Systems Consultants, Inc.
™ CP/M-68K is trademark of Digital Research, Inc.

tFORTH™
from TALBOT MICROSYSTEMS
NEW SYSTEMS FOR
6301/6801, 6809, and 68000

---> tFORTH SYSTEMS <---

For all FLEX systems: GIMIX, SWTP, SSB, or EXORcisor Specify 5 or 8 inch diskette, hardware type, and 6800 or 6809.

** tFORTH — extended fig FORTH (1 disk) $100 ($15)
with fig line editor.
** tFORTH+ — more! (3 5" or 2 8" disks) $250 ($25)
adds screen editor, assembler, extended data types, utilities, games, and debugging aids.
** TRS-80 COLORFORTH — available from The Micro Works
** firm FORTH — 6809 only. $350 ($10)
For target compilations to rommable code. Automatically deletes unused code. Includes HOST system source and target nucleus source. No royalty on targets. Requires but does not incl de tFORTH+.
** FORTH PROGRAMMING AIDS — elaborate decompiler $150

** tFORTH for HX-20, in 16K roms for expansion unit or replace BASIC $170

** tFORTH/68K for CP/M-68K 8" disk system $290
Makes Model 16 a super software development system.

** Nautilus Systems Cross Compiler
— Requires: tFORTH + HOST + at least one TARGET:
— HOST system code (6809 or 68000) $200
— TARGET source code: 6800-$200, 6301/6801—$200
same plus HX-20 extensions— $300
6809—$300, 8080/Z80— $200, 68000—$350

Manuals available separately — price in ( ).
Add $6/system for shipping, $15 for foreign air.

TALBOT MICROSYSTEMS 1927 Curtis Ave., Redondo Beach, CA 90278 (213) 376 9941

WINDRUSH MICRO SYSTEMS

UPROM II

PROGRAMS and VERIFIES: I2758, T2508, I2716, T2516, I2732/2732A, MCM68764/6, I2764/2764A, T2564, I27128/27128A, and I27256. I=Intel, T=Texas, M=Motorola.

NO PERSONALITY MODULES REQUIRED!

TRI-VOLT EPROMS ARE NOT SUPPORTED

INTEL's intelligent programming (tm) implemented for Intel 2764, 27128 and 27256 devices. Intelligent programming reduces the average programming time of a 2764 from 7 minutes to 1 minute 15 seconds (under FLEX) with greatly improved reliability.

Fully enclosed pod with 5' of flat ribbon cable for connection to the host computer MC6821 PIA Interface board.

MC6809 software for FLEX and OS9 (Level 1 or 2, Version 1.2).

BINARY DISK FILE offset loader supplied with FLEX, MDOS and OS9.

Menu driven software provides the following facilities:

a.	FILL	a selected area of the buffer with a HEX char.
b.	MOVE	blocks of data.
c.	DUMP	the buffer in HEX and ASCII.
d.	FIND	a string of bytes in the buffer.
e.	EXAMINE/CHANGE .	the contents of the buffer.
f.	CRC	checksum a selected area of the buffer.
g.	COPY	a selected area of an EPROM into the buffer.
h.	VERIFY	a selected area of an EPROM against the buffer.
i.	PROGRAM	a selected area of an EPROM with data in the buffer.
j.	SELECT	a new EPROM type (return to types menu).
k.	ENTER	the system monitor.
l	RETURN	to the operating system.
l.	EXECUTE	any DOS utility (only in FLEX and OS9 versions).

FLEX AND OS9 VERSIONS AVAILABLE FROM GIMIX. SSB/MDOS CONTACT US DIRECT.

MACE/XMACE/ASM05

All of these products feature a highly productive environment where the editor and the assembler reside in memory together. Gone are the days of tedius disk load and save operations while you are debugging your code.

- Friendly inter-active environment where you have instant access to the Editor and the Assembler, FLEX utilities and your system monitor.
- MACE can also produce ASMPROCs (GEN state mts) for PL/9 with the assembly language source passed to the output as comments.
- XMACE is a cross assembler for the 6800/1/2/3/8 and supports the extended mnemonics of the 6303.
- ASM05 is a cross assembler for the 6805.

D-BUG

LOOKING for a single step trater and mini in-line disassembler that is easy to use?? Look no further, you have found it. This package is ideal for those small assembly language program debugging sessions. D-BUG occupies less than 6k (including its stack and variables) and may be loaded anywhere in memory. All you do is LOAD it, AIM it and GO! (80 col VDUs only).

McCOSH 'C'

This is as complete a 'C' compiler as you will find on any operating system for the 6809. It is completely compatible with UNIX VII and only lacks 'bit-fields' (which are of little practical use in an 8-bit world!).

- Produces very efficent assembly language source output with the 'C' source optionally interleaved as comments.
- Built-in optimizer will shorten object code by about 11%.
- Supports interleaved assembly language programs.
- INCLUDES its own assembler. The TSC relocating assembler is only required if you want to generate your own libraries.
- The pre-processor, compiler, optimizer, assembler and loader all run independently or under the 'CC' executive. 'CC' makes compiling a program to executable object as simple as typing in 'CC,HELLO.C <RETURN>'.

PL/9

- Friendly inter-active environment where you have INSTANT access to the Editor, the Compiler, and the trace-debugger, which, amongst other things, can single step the program a SOURCE line at a time. You also have direct access to any FLEX utility and your system monitor.
- 375+ page manual organized as a tutorial with plenty of examples.
- Fast SINGLE PASS compiler produces 8K of COMPACT and FAST 6809 machine code output per minute with no run-time overheads or license fees.
- Fully compatible with TSC text editor format disk files.
- Signed and unsigned BYTES and INTEGERS, 32-bit floating point REALS.
- Vectors (single dimension arrays) and pointers are supported.
- Mathematical expressions: (+), (-), (*), (/), modulus (\), negation (-)
- Expression evaluators: (=), (<>), (<), (>), (>=), (<=)
- Bit operators: (AND), (OR), (EOR/XOR), (NOT), (SHIFT), (SWAP)
- Logical operators: (.AND), (.OR), (.EOR/XOR)
- Control statements: IF..THEN..ELSE, IF..CASE1..CASE2..ELSE, BEGIN..END, WHILE.., REPEAT..UNTIL, REPEAT..FOREVER, CALL, JUMP, RETURN, BREAK, GOTO.
- Direct access to (ACCA), (ACCB), (ACCD), (XREG), (CCR) and (STACK).
- FULLY supports the MC6809 RESET, NMI, FIRQ, IRQ, SWI, SWI2, and SWI3 vectors. Writing a self-starting (from power-up) program that uses ANY, or ALL, of the MC6809 interrupts is an absolute snap!
- Machine code may be embedded in the program via the 'GEN' statement. This enables you to code critical routines in assembly language and embed them in the PL/9 program (see 'MACE' for details).
- Procedures may be passed and may return variables. This makes them functions which behave as though they were an integral part of PL/9.
- Several fully documented library procedure modules are supplied: IOSUBS, BITIO, HARDIO, HEXIO, FLEXIO, SCIPACK, STRSUBS, BASTRING, and REALCON.

"... THIS IS THE MOST EFFICIENT COMPILER I HAVE FOUND TO DATE."

Quoted from Ron Andersons FLEX User Notes column in '68. Need we say more?

IEEE-488

- SUPPORTS ALL PRINCIPAL MODES OF THE IEEE-488 (1975/8) BUS SPECIFICATION:

- Talker	- Serial Poll	- Single or Dual Primary Address
- Listener	- Parallel Poll	- Secondary Address
- System Controller	- Group Trigger	- Talk only ... Listen only

- Fully documented with a complete reprint of the KILOBAUD article on the IEEE bus and the Motorola publication 'Getting aboard the IEEE Bus'.
- Low level assembly language drivers suitable for 6800, 6801, 6802, 6803, 6808 and 6809 are supplied in the form of listings. A complete back to back test program is also supplied in the form of a listing. These drivers have been extensively tested and are GUARANTEED to work.
- Single S-30 board (4, 8 or 16 addresses per port), fully socketed, gold plated bus connectors and IEEE interface cable assembly.

PRICES

D-BUG	(6809 FLEX only)	$ 75.00
MACE	(6809 FLEX only)	$ 75.00
XMACE	(6809 FLEX only)	$ 98.00
ASM05	(6809 FLEX only)	$ 98.00
PL/9	(6809 FLEX only)	$198.00
'C'	(6809 FLEX only)	$295.00
IEEE-488	with IEEE-488 cable assembly	$298.00
UPROM-II/U	with one version of software (no cable or interface) ..	$395.00
UPROM-II/C	as above but complete with cable and S-30 interface	$545.00
CABLE	5' twist-n-flat 50 way cable with IDC connectors	$ 35.00
S-30 INT	SS-30 interface for UPROM-II	$130.00
EXOR INT	Motorola EXORbus (EXORciser) interface for UPROM-II ...	$195.00
UPROM SFT	Software drivers for 2nd operating system. Specify FLEX or OS9 AND disk size!	$ 35.00
UPROM SRC	Assembly language source (contact us direct)	

A L L   P R I C E S   I N C L U D E   A I R   M A I L   P O S T A G E

Terms: CWO. Payment by Int'l Money Order. VISA or MASTER-CARD also accepted.

WORSTEAD LABORATORIES, NORTH WALSHAM, NORFOLK, ENGLAND. NR28 9SA.
TEL: 44 (692) 404086
TLX: 975546 WMICRO G

WE STOCK THE FOLLOWING COMPANIES PRODUCTS: GIMIX, SSB, FHL, MICROWARE, TSC, LUCIDATA, LLOYD I/O, & ALFORD & ASSOCIATES.

FLEX (tm) is a trademark of Technical Systems Consultants, OS-9 (tm) is a trademark of Microware Systems Corporation, MDOS (tm) and EXORciser (tm) are trademarks of Motorola Incorporated.

'68' MICRO JOURNAL

OK, PLEASE ENTER MY SUBSCRIPTION

Bill My: Master Charge ☐ — VISA ☐

Card # ____________ Exp. Date ____________

For ☐ 1-Year ☐ 2 Years ☐ 3 Years

Enclosed: $ ____________

Name ____________

Street ____________

City ____________ State ____________ Zip ____________

My Computer Is: ____________

Subscription Rates
(Effective March 3, 1985)

U.S.A.: 1 Year $24.50, 2 Years $42.50, 3 Years $64.50

*** Foreign Surface: Add $12.00 per Year to USA Price.**

*** Foreign Airmail: Add $48.00 per Year to USA Price.**

*** Canada & Mexico: Add $ 9.50 per Year to USA Price.**

*** U.S. Currency Cash or Check Drawn on a USA Bank !!!**

68 Micro Journal
5900 Cassandra Smith Rd.
Hixson, TN 37343

VISA

(615)842-4600

Telex 5106006630

MasterCard

ATTENTION all STAR-DOS Users!

We have made some very significant changes, improvements, and additions to STAR-DOS™.

STAR-DOS now ... handles random files with unsurpassed speed ... even allows random files to be extended ... loads programs faster than ever ... has extensive error checking to avoid operator and system errors ... comes with source code to allow you to modify STAR-DOS for your system to add automatic date insertion, time stamping of files (even random files), and RAM disks up to a megabyte ... allows up to ten drives ... includes a wide selection of utilities which often cost extra on other systems ... comes with superb documentation and user support.

We greatly suggest that you update your present copy of STAR-DOS. Just send us your original STAR-DOS disk along with $3 to cover postage, handling, and a 15-page update manual.

And if you aren't using STAR-DOS yet

why not switch from FLEX (tm of Technical Systems Consultants) to STAR-DOS (our trademark) now? Consider also our SPELL 'N FIX and MAGIC SPELL spelling correction programs, WRITE 'N SPELL dictionary lookup program, HUMBUG monitor and debug system, CHECK 'N TAX home accounting system, SBC-02-B single-board control computer, and more. Write for a catalog, or call us at (914) 241-0287.

STAR-KITS SOFTWARE SYSTEMS CORPORATION Box 209 Mt. Kisco NY 10549

ANDERSON COMPUTER CONSULTANTS
&
Associates

Ron Anderson, respected author and columnist for 68 MICRO JOURNAL announces the **Anderson Computer Consultants & Associates**, a consulting firm dealing primarily in 68XX(X) software design. Our wide experience in designing 6809 based control systems for machine tools is now available on a consultation basis.

Our experience includes programming machine control functions, signal analysis, multi-axis servo control (CNC) and general software design and development. We have extensive experience in instrumentation and analysis of specialized software. We support all popular languages pertaining to the 6809 and other 68XX(X) processors.

If you are a manufacturer of a control or measuring package that you believe could benefit from efficient software, write or call Ron Anderson. The fact that any calculation you can do with pencil and paper, can be done much better with a microcomputer. We will be happy to review your problem and offer a modern, state-of-the-art microcomputer solution. We can do the entire job or work with your software or hardware engineers.

Anderson Computer Consultants & Associates
3540 Sturbridge Court
Ann Arbor, MI 48105

INDUSTRIAL PASCAL FOR THE 68000

If you're looking for a language to write real-time process control software, look no further. With the rising cost of labor, it is becoming critical that a high level language be used whenever possible. Find out why over 1400 companies have switched to OmegaSoft Pascal for their demanding applications.

OmegaSoft Pascal takes the Pascal framework and expands the basic data types, operators, functions, and memory allocation to fit the needs of real-time systems. These additions fit in the same structure as Pascal and enhance its usefulness without impairing the excellent readability, ease of maintenance, and structured design.

The compiler generates assembly language for assembly and link to run on the target system. Since a true relocating assembler and linking loader is used, only those runtime modules required are automatically linked in, providing a smaller object module than other compilers.

Large Pascal programs can be split up into conveniently sized modules to speed the development process. Procedures, functions, and variables can be referenced between Pascal modules and assembly language modules by using Pascal directives.

The compiler package includes an interactive, symbolic debugger. The debugger allows setting of breakpoints, displaying and changing variables, and tracing statements. Debugging can also be done at the assembly language level when needed. The debugger allows very fast turnaround for programs to be run on the host system (target system debugger coming soon).

The compiler package also includes a full relocatable macro assembler and linking loader. These are designed to support the compiler but may also be used for general assembly language development. In addition, a full screen editor is included which can be used with a variety of intelligent terminals.

Full source code is included for the runtime library, the debugger, the screen editor, and other support utilities.

Versions to run under the OS-9/68000 and VERSAdos operating systems are currently available to end-users and OEM's. End user price is $900 (domestic) or $925 (international). A version for CP/M-68K is available for OEM use, with OEM versions for UNIX type operating systems to follow.

Similar products to run on a 6809 system and generate 6809 code are also available for most major 6809 operating systems.

T.M. OmegaSoft is a trademark of Certified Software Corporation. OS-9/68000 is a trademark of Microware. VERSAdos is a trademark of Motorola. CP/M-68K is a trademark of DRI. UNIX is a trademark of Bell Labs.

CERTIFIED SOFTWARE CORPORATION

616 Camino Caballo, Nipomo, CA 93444
Telephone: (805) 929-1395; Telex: 467013

DRIVE ENCLOSURES

FLOPPY 8" & 5"

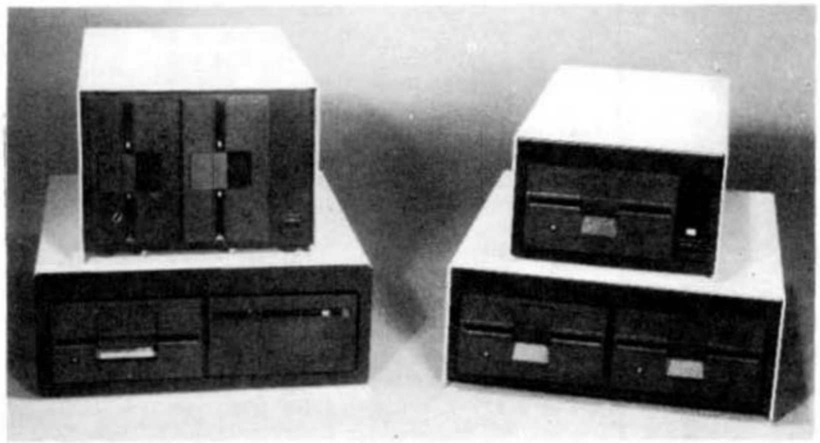
TAPE 5" & 8"

WINCHESTER 5" & 8"

FLOPPY-WINCHESTER-TAPE

FROM $80⁰⁰

(Includes Power Supply)



- Desktop & Rack
- Heavy Duty All Metal Cabinet
- Fan & Dust Filter*
- Hefty Power Supplies
- Full or Slim Drives
- Power Harness From Supply To Drives
- Line Fuse, EMI Filter*, Detachable Line Cord
- Cabinets & Supplies Available Separately

* = Most Models (Disk drives not included)

INTEGRAND

RESEARCH CORPORATION

8620 Roosevelt Ave./Visalia, CA 93291

209/651-1203

Write or call for our brochure which includes our application note: "Making micros. better than any ol' box computer"

32 Page Free Fakt Pakt Catalog

OS-9

USERS GROUP

...Information Exchange

...Software Library over 40 diskettes

...Message Of The Day periodic newsletter

Write or Go OS9 on CompuServe for information

OS-9 Users Group
P.O. Box 7586
Des Moines, IA 50322

OS-9 is a trademark of Microware, Inc.
The Users Group is not affiliated with Microware

Hard Disk Subsystem for SS-50 Computers

This proven subsystem adds hard disk speed and storage capacity to your computer yet requires only one SS 30 slot. Software (with source) is included for your choice of FLEX9*, OS-9* Level I or Level II, or OS-9 68K operating systems. The software honors all operating system conventions. The software is designed for the Xebex S1410 controller interfacing to any hard disk drive that conforms to the ST506 standard. Four subsystems are available:

1) 27 MB (formatted) WREN* hard disk, Xebec S1410 controller, SS-30 interface card, all cables, and software for $2850;
2) 5 MB (formatted) Shugart 604 hard disk, rest same as above for $750;
3) no hard disk, rest same as above for $600; and
4) SS-30 interface card and software for $200.

Texas residents must add sales tax. The subsystem may be mounted within your computer chassis or in a seperate enclosure with power supply. Please write or phone (include your day and evening phone numbers) for more information. We will return North America calls so that any detailed answers will be at our expense. We are proud to announce our relocation to the growing High-Tech center of the Southwest--Austin. Our new phone number is (512) 335-9446. We are now able to accept payment by Visa and Mastercard.

WELLWRITTEN™ ENTERPRISES

M/C VISA

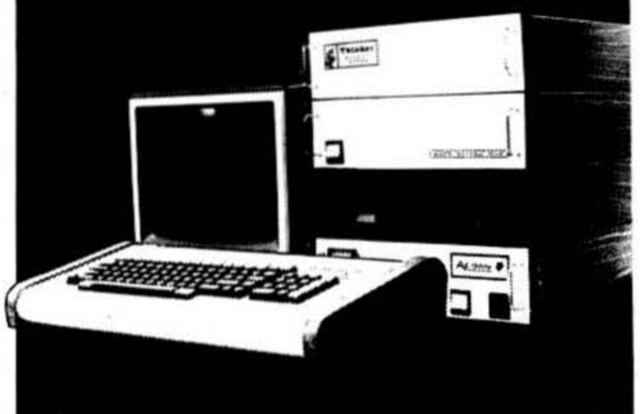
P.O. Box 9802 - 845
Austin, Texas 78766

Phone
512 335-9446

FLEX is a trademark of Technical Systems Consultants, Inc.
OS-9 is a trademark of Microware and Motorola
WREN is a trademark of Control Data Corporation

ACORN

COMPUTER SYSTEMS SS-50C



MODULES - BARE CARDS - KITS - ASSEMBLED & TESTED

Stackable Modules	KIT	A&T
20 amp POWER SUPPLY w/fan w/Disk protect relay	350.00	400.00
DISK CABINET w/regs. & cables less DRIVES	200.00	250.00
MOTHER BOARD, 8 SS-50c, 8 SS-30c NMI button	225.00	325.00

Item	Bare	KIT	A&T
IT3 - INTERRUPT TIMER 1, 10, 100 per sec.	19.95	29.95	39.95
PB4 - INTELLIGENT PORT BUFFER Single board comput.	39.95	114.95	139.95
DPIA - Dual PIA parallel port, 4 buffered I/Os	24.95	69.95	89.95
XADR - Extended Addressing BAUD gen. PIA port	29.95	69.95	89.95
MB8 - MOTHER BOARD SS-50c w/BAUD gen.	64.95	149.95	199.95
P168 - 168K PROM DISK 21, 2764 EPROMs	39.95	79.95	109.95
FD68 - Firmware development 2, 8K blocks	39.95	84.95	114.95
XMPR - 2764 PROM burner adapt. for 2716 BURNER		19.95	------
CHERRY Keyboard w/Cabinet 98 key capacitive		249.95	------
TAXAN 12", 18 Mhz MONITOR GREEN		------	149.95
AMBER		------	159.95
4 MODULE CABINET - unfinished		150.00	------
POWER SUPPLY w/disk protect		250.00	------

Color Computer

MONOLINK - 20 Mhz Monochrome video driver	15.00	20.00
CC30 PORT BUS w/power supply 5 SS-30, 2 Cart	169.95	199.95
POWER BOX 6 switched outlets transient suppression	29.95	39.95
RS-232 3-switched ports for above	ADD +20.00	+25.00

Write for FREE Catalog
ADD $5.00 S&H PER ORDER
WIS. ADD 5% SALES TAX

VISA

ACS INC

11931 W. Bluemound Road
MILWAUKEE, WIS. 53226
(414) 257-0300

68' MICRO JOURNAL

Disk-1 Filesort, Minicat, Minicopy, Minifms, **Lifetime, **Poetry, **Foodlist, **Diet.
Disk-2 Diskedit w/ inst.& fixes, Prime, *Prmod, **Snoopy, **Football, **Hexpawn, **Lifetime
Disk-3 Cbug09, Sec1, Sec2, Find, Table2, Intext, Disk-exp, *Disksave.
Disk-4 Mailing Program, *Finddat, *Change, *Testdisk.
DISK-5 *DISKFIX 1, *DISKFIX 2, **LETTER, **LOVESIGN, **BLACKJAK, **BOWLING.
Disk-6 **Purchase Order, Index (Disk file indx)
Disk-7 Linking Loader, Rload, Harkness
Disk-8 Crtest, Lanpher (May 82)
Disk-9 Datecopy, Diskfix9 (Aug 82)
Disk-10 Home Accounting (July 82)
Disk-11 Dissembler (June 84)
Disk-12 Modem68 (May 84)
Disk-13 *Initmf68, Testmf68, *Cleanup, *Dskalign, Help, Date.Txt
Disk-14 *Init, *Test, *Terminal, *Find, *Diskedit, Init.Lib
Disk-15 Modem9 + Updates (Dec. 84 Gilchrist) to Modem9 (April 84 Commo)
Disk-16 Copy.Txt, Copy.Doc, Cat.Txt, Cat.Doc
Disk-17 Match Utility, RATBAS, A Basic Preprocessor
Disk-18 Parse.Mod, Size.Cmd (Sept. 85 Armstrong), CMDCODE, CMD.Txt (Sept. 85 Spray)
Disk-19 Clock, Data, Copy, Cat, PDEL.Asm & Doc., Errors.Sys, Do, Log.Asm & Doc.
Disk-20 UNIX Like Tools (July & Sept. 85 Taylor & Gilchrist), Dragon.C, Grep.C, LS.C, FDUMP.C
Disk-21 Utilities & Games - Date, Life, Madness, Touch, Goblin, Starshot, & 15 more.
Disk-22 Read CPM & Non-FLEX Disks. Fraser May 1984.
Disk-23 ISAM, indexed sequential file accessing methods. Condon Nov. 1985.

NOTE:
This is a reader service ONLY! No Warranty is offered or implied, they are as received by '68' Micro Journal, and are for reader convenience ONLY (some MAY include fixes or patches). Also 6800 and 6809 programs are mixed, as each is fairly simple (mostly) to convert to the other.

PRICE: 8" Disk $14.95 - 5" Disk $12.95

68' MICRO JOURNAL
POB 794
Hixson, TN 37343
615-842-4600

* Indicates 6800
** Indicates BASIC SWTPC or TSC
6809 no Indicator.

MASTER CARD - VISA Accepted
Foreign -- add **10%** for Surface
or **20%** for Air!!

VISA

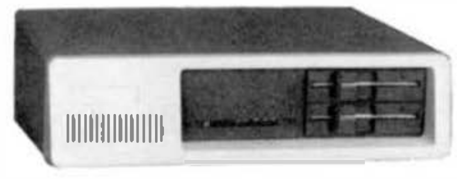
Telex 5106006630

PT-69 SINGLE BOARD COMPUTER SYSTEMS

NOW WITH WINCHESTER OR FLOPPY DISK DRIVES

The proven PT-69 Single Board Computer line is expanding! Systems now can be Winchester or floppy-based. Available also in a smaller cabinet without drives for dedicated systems with no mass storage requirements.

- 1 MHZ 6809E Processor
- Time-of-Day Clock
- 2 RS 232 Serial Ports (6850)
- 56K RAM 2K/4K EPROM
- 2 8-bit Parallel Ports (6821)
- 2797 Floppy Disk Controller


Winchester System

Custom Design Inquiries Welcome

Floppy System

• PT69XT WINCHESTER SYSTEM Includes 5 MEG Winchester Drive, 2 40-track DS/DD Drives, Parallel Printer Interface + choice of OS/9 or STAR-DOS	$1795.95
• PT6952 FLOPPY SYSTEM Includes PT69 Board, 2 DS/DD 40-TRK 5 1/4" drives, cabinet, switching power supply, OS/9 or STAR-DOS	$895.95

• PT-69A ASSEMBLED & TESTED BOARD	$279.00
• OS/9	$200.00
• STAR-DOS	$ 50.00

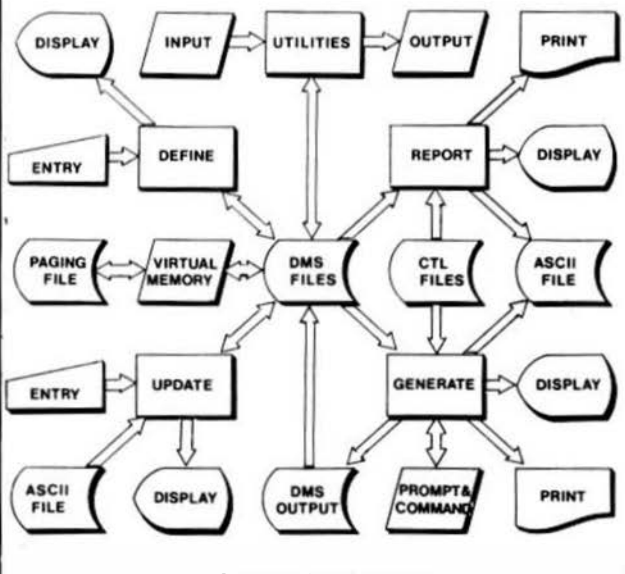
PERIPHERAL TECHNOLOGY
1480 Terrell Mill Rd., Suite 870
Marietta, Georgia 30067

VISA/MASTERCARD/CHECK/COD   Telex #880584   404/973-0042   CALL OR WRITE FOR ADDITIONAL CONFIGURATIONS

[illegible]

XDMS

Data Management System



System Architecture

WESTCHESTER Applied Business Systems
Post Office Box 187
Briarcliff Manor, N.Y. 10510

XDMS Data Management System

The XDMS Data Management System is available in three levels. Each level includes the XDMS nucleus, VMOEX utility and System Documentation for level III. XDMS is one of the most powerful systems available for 6809 computers and may be used for a wide variety of applications. XDMS users are registered in our database to permit distribution of product announcements and validation of user upgrades and maintenance requests.

XDMS Level I

XDMS Level I consists of DEFINE, UPDATE and REPORT facilities. This level is intended as an "entry level" system, and permits entry and reporting of data on a "tabular" basis. The REPORT facility supports record and field selection, field merge, sorting, line calculations, column totals and report titling. Control is via a English-like language which is upward compatible with level II. XDMS Level I $129.95

XDMS Level II

Level II adds to Level I the powerful GENERATE facility. This facility can be thought of as a general file processor which can produce reports, forms and form letters as well as file output which may be re-input to the facility. GENERATE may be used in complex processing applications and is controlled by a English-like command language which encompasses that used by Level I. XDMS Level II $179.95

XDMS Level III

Level III includes all of level II plus a set of useful DMS Utilities. These utilities are designed to aid in the development and maintenance of user applications and permit modification of XDMS system parameters, input and output of XDMS files, display and modification of file format, graphic display of numerical data and other functions. Level III is intended for advanced XDMS users. XDMS Level III $269.95
XDMS System Documentation only (a10. credit toward purchase). . . $ 24.95

XACC Accounting System

The XACC General Accounting System is designed for small business environments of up to 10,000 accounts and inventory items. The system integrates accounting functions and inventory plus the general ledger, accounts receivable and payable functions normally sold separately in other systems. Features user defined accounts, products (or services), transactions, invoicing, etc. Easily configured to most environments.
XACC General Accounting System (Requires XDMS, pref. Lv. III). . $299.95
XACC System Documentation only (a10. credit toward purchase). . . $ 24.95

WESTCHESTER Applied Business Systems
Post Office Box 187, Briarcliff Manor, N.Y. 10510

All software is written in macro/assembler and runs under 6809 FLEX O/S. Terms: Check, Money Order, Visa or Mastercharge. Shipment first class. Add P&H $2.50 ($7.50 Foreign Surface or $15.00 Foreign Air). NY Res add sales tax. Specify 5" or 8".

Sales: S. E. MEDIA, (615) 842-4600, Consultation: 914-941-3552 (evens)

GMX 68020 DEVELOPMENT SYSTEM

A Multi-user, Multi-tasking software development system for use with all 68000 family processors.

HARDWARE FEATURES:

- The GMX-020 CPU board has: the MC68020 32-bit processor, a 4K Byte no wait-state instruction cache, high-speed MMU, and a full-featured hardware time of day clock/calendar with battery back-up. It also provides for an optional 68881 floating point co-processor.
- 1 Megabyte of high speed static RAM.
- Intelligent Serial and Parallel I/O Processor boards significantly reduce system overhead by handling routine I/O functions. This frees up the host CPU for running user programs. The result is a speed up of system performance and allows all terminals to run at up to 19.2K baud.
- The system hardware will support up to 39 terminals.
- Powered by a constant voltage ferro-resonant power supply that insures proper system operation under adverse AC power input conditions.
- DMA hard disk interface and DMA double density floppy disk controller are used for data transfers at full bus speed. The DMA hard disk drive controller provides automatic 22-bit burst data error detection and 11-bit burst error correction.
- A selection of hard disk drives with capacities from 19 to 85 Megabytes, removeable pack hard disk drives, streaming tape drives, and floppy disk drives is available.

UNIX is a trademark of A.T. & T.
ADA is a trademark of the U.S. Government
UniFLEX is a trademark of Technical Systems Consultants, Inc.
GMX and GIMIX are trademarks of GIMIX, Inc.

SOFTWARE FEATURES:

The UniFLEX VM Operating System is a demand-paged, virtual memory operating system written in 68020 Assembler code for com-pactness and efficiency. Any UniFLEX system will run faster than a comparable system written in a higher level language. This is impor-tant in such areas as context switching, disk I/O, and system call handling. Other features include:

- compact, efficient Kernel and modules allows handling more users more effectively than UNIX systems, using much less disk space.
- UNIX system V compatibility at the C source code level.
- C Compiler optimized in 68020 code (optional).
- Record locking for shared files.
- Users can share programs in memory.
- Modeled after UNIX systems, with similar commands.
- System accounting facilities.
- Sequential and random file access.
- Maximum record size limited only by the disk size.
- Multiple Level Directories.
- Up to 4 Megabytes of Virtual Memory per user.
- Optional Languages available are: C, BASIC, COBOL, FORTRAN, LISP, PROLOG, SCULPTOR, and ASSEMBLER. In development ar ADA, PASCAL, and FORTH.

Included with the UniFLEX Operating System are a Utilities package editor, relocating assembler, linking loader, and printer spooler. Op tions inlcude a fast floating point package, library generator, and sort-merge package.

The GMX version of the MOTOROLA 020 BUG is included with th system.

GIMIX, Inc., a Chicago based microcomputer company established in 1975, has produced state of the art microcomputer systems based on Motorola 6800 an 6809 microprocessors. GIMIX systems are in use in Industry,, Hospitals, Universities, Research Organiations, and by Software Developers. GIMIX was awarde the prestigious President's "E" Certificate for Exports in 1984.

GIMIX inc. 1337 WEST 37th PLACE • CHICAGO, ILLINOIS 60609 • (312) 927-5510 • TWX910-221-405

Southwest Technical Products Announces: Building Block Systems

Dealer Inquiries Invited

S1000 System
Printer & Terminal Optional

Engineered for Expansion

Southwest provides two families of Building Block Systems carefully engineered to meet the demanding requirements of business and industrial environments. The Building Block Systems are the starting point for building a system tailored to your specific application.

Solutions Hardware

Each system is configured with a powerful Motorola 68809 Central Processing Unit, mass storage and expansion ports for adding additional devices.

- All the Building Block Systems come equipped with one or two 5¼ inch floppy disk drives. Each floppy disk drive has a 1.25 million character capacity.
- A variety of Winchester disk drives are available for applications with large data volumes and/or multiple users.
- The S1000 and S1100 are delivered in an attractive office environment system cabinet with casters for easy movement. All other Building Block Systems come as two stackable boxes.
- The Building Block Systems come ready to accept at least one terminal and one printer. Larger systems are configured to support additional peripheral devices.

Solutions Software

The S600 system includes the versatile single-user FLEX™ Operating System. Requiring less than 56K bytes of memory, FLEX™ is the operating system of choice for single-user applications. FLEX™ developers have a choice of Assembler, BASIC, C and Fortran for developing applications. A variety of system utilities are also available.

All other systems are shipped with UniFLEX®.

UniFLEX® (UNIX™ look-alike) provides a powerful multi-user, multi-tasking operating system. Residing in less than 60,000 bytes of memory, UniFLEX® is capable of supporting large installations of up to sixteen terminals and multiple printers. The UniFLEX® operating system supports Assembler, BASIC, C, COBOL, Fortran and Pascal. A variety of system utilities are also available.

Building Blocks

Each of the Building Block Systems is ready for the user to expand as needed. In general, the S600 and S650 are intended for applications which need four or less users on-line. The S900 and S950 are targeted for applications with four to eight terminals. Both the S1000 and the S1100 can easily support up to sixteen terminals and provide for maximum growth potential.

Model	CPU Memory	Floppy Disk	Winchester Disk	Tape Drive	Number Of Ports	List Price
S600	64Kb	2			3	2750
S650	256Kb	1	26.7Mb		5	5700
S900	256Kb	1	26.7Mb		5	6000
S950	512Kb	1	40Mb	40Mb	5	9000
S1000	512Kb	1	51Mb	40Mb	9	11,500
S1100	512Kb	1	85Mb	40Mb	9	13,000

UNIX™ is a trademark of AT&T Bell Laboratories. UniFLEX® and FLEX™ are registered trademarks of Technical Systems Consultants, Inc.

Specifications and prices are subject to change without notice.

Southwest Technical Products, Corp.
219 West Rhapsody • San Antonio, Texas 78219
(512) 344-0241

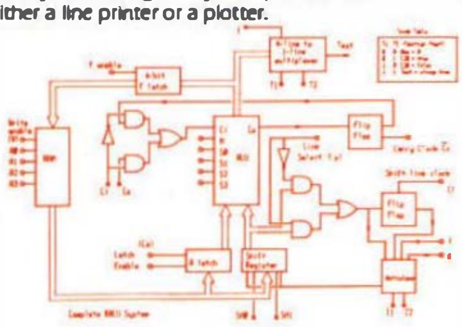
Southwest Technical Products Announces:

CAD Southwest A Computer-Aided Drafting System

Dealer Inquiries Invited

Designed for the Professional!

Southwest's Computer-Aided Drafting System represents the state of the art in computer-aided drafting. It eliminates the need for labor intensive drafting using the traditional T-square and triangle approach. Via keyboard or a sophisticated digitizer tablet, you can quickly and easily prepare even the most complicated drawings. Naturally, changes can be easily made at any time. A library of standard symbols is provided as well as the flexibility of a user defined symbol library. Drawings may be printed on either a line printer or a plotter.

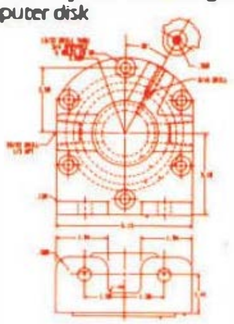


Increased Productivity

A powerful set of commands allows even the novice user the ability to quickly and easily prepare complex drawings. The experienced draftsman will quickly realize substantial improvements in productivity. The computer assumes the entire burden of line and arc drawing, freeing the user to concentrate on design issues. A partial list of key features includes:

- no limit to number of drawings
- standard grid pattern, used for reference; will increase or decrease with scale
- isometric grid pattern - similar to standard
- library mode - complete objects can be drawn and stored in users' library for later use in any drawing
- mirror objects - images of objects in library can be easily duplicated
- the terminal screen acts as a window by which the user can quickly move large drawings
- draw/delete functions:
  - arc/fillets (solid or dashed)
  - circles (solid or dashed)
  - lines and sequence of lines (solid or dashed)
  - true horizontal/vertical lines
  - center lines
  - dimension lines
  - extension lines
  - dimension lines perpendicular to extension lines
  - extension lines perpendicular to normal lines
  - boxes: any shape or size defined by two diagonal endpoints
  - library objects: previously defined library objects drawn or deleted with a couple of keystrokes
    - area fills
    - special functions:
      - change line mode: solid to dashed, dashed to solid
      - sector circle: save portion of circle as arc
      - segment lines: break single line into two lines
    - scale using factor - choose desired scale multiple
    - scale using window - "zoom in" and work on a selected area
- complete set of labels - comprehensive label generation component to annotated drawings
- overlays - any drawing can have up to 255 overlays
- cursor functions:
  - variable size: user can scale cursor to convenient size
  - variable step: user can select single pixel/giant step for movement of cursor
  - homing: cursor can be quickly moved to 9 possible predefined home positions on screen
- complete disk-oriented library maintains routines that allow you to maintain your drawings on the computer disk

ARCHITECTURAL DESIGN USING CAD-SOUTHWEST DRAFTING SYSTEM



Big System Features

As you prepare your drawing, you can control which portion is available for work on the terminal screen. Simple commands allow you to move up, down and sideways to view different segments of a large drawing. "CAD ZOOM" allows you to quickly "zoom in," turning your terminal into a computerized magnifying glass for detailed review of individual sections. "CAD ZOOM" compresses the drawing, allowing you to look at multiple sets of the drawing on one screen.

A powerful overlay feature supports up to 255 overlays for the same drawing. For example, you may wish to have the floor plan on one overlay, electrical wiring on a second, HVAC on a third, and so forth.

The Right Equipment

CAD Southwest runs on the proven SWTPc family of computers operating under UniFLEX®, a UNIX™ like operating system. Five to eighteen terminals, depending on the model number, can be attached to the CPU with a variety of hard disk storage for your drawings.

The industrial quality X-12 terminal from Southwest provides crisp resolution and a 92-key keyboard. A digitizing tablet may be optionally attached to each terminal.

Drawings may be printed on a dot matrix printer or optionally on an 8 color pen plotter.

UNIX™ is a trademark of AT&T Bell Laboratories. UniFLEX® is a registered trademark of Technical Systems Consultants, Inc.

Southwest Technical Products, Corp.
219 West Rhapsody • San Antonio, Texas 78216
(512) 344-0241